



44506

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Application of : **KALLNER et al.**

Serial No.: 10/053,872

:

: Group Art Unit: 2142

:

Filed : January 24, 2002 : Examiner: Benjamin A. Ailes

:

For : COMMUNICATION ENDPOINT SUPPORTING MULTIPLE
PROVIDER MODELS

Honorable Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

DECLARATION UNDER 37 CFR 1.131

Sir:

We, the undersigned, Samuel Kallner, Lev Kozakov, Alexey Roytman, Uri Shani, and Pnina Vortman, hereby declare as follows:

1) We are the Applicants in the patent application identified above, and are the inventors of the subject matter described and claimed in claims 1, 4-16, 26, 29-41, 56 and 59-71 therein.

2) Prior to September 6, 2000, we reduced our invention to practice, as described and claimed in the subject application, in Israel, a WTO country. We implemented the invention in the form of software code in the Java programming language. This code provided an application programming interface (API),

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

which enabled calls to be connected between parties via different service providers with different telephony signaling stacks, using an abstract call model, as recited in the claims of this patent application. The software code was tested in placing actual telephone calls, as described further hereinbelow.

3) As evidence of the reduction to practice of the present invention, we attach hereto in Exhibit A selected portions of the software source code that we used to implement the invention. A directory listing in Exhibit B (generated by the CMVC configuration management version control system used in the IBM Haifa Research Laboratory) shows the dates on which the source code files were stored on disk. The dates of the files, which are blacked out in Exhibits A and B, are all prior to September 6, 2000.

4) Generally speaking, the attached software is a special implementation of the JTAPI specification, which we referred to as "Generic JTAPI" (or GenJTAPI). The GenJTAPI implementation provides an abstract, platform-independent call model with several interfaces (as shown in Fig. 2 of the patent application):

- a standard JTAPI interface for interaction with telephony applications;
- a Java Telephony Service Provider Interface (JTSPI) for interaction with service provider plug-ins; and
- a Java Telephony Service Management Interface (JTSMI) for interaction with a management plug-in, which receives call information, such as the telephone number of a called party, and returns instructions regarding selection of appropriate plug-ins.

Declaration under 37 C.F.R 1.131 by Kallner et al.

For each party participating in a call, the GenJtapiPeer class in Exhibit A supplies a hybrid JTAPI provider instance, which interacts with the appropriate service provider plug-in via the JTSPi. In this manner, parties served by different service providers, with different telephony stacks, can be connected and communicate in a call initiated by a telephony application via the generic JTAPI layer.

5) When an application first creates an instance of the GenJtapiPeer class, an initialization process is started. This process includes reading configuration files and creating a registry and other data structures that reflect registered service providers and resources. After the initialization process is finished, when the application initiates a call, the management (JTSMI) plug-in looks up the dialed number and uses it to identify the appropriate service provider for the called party. The application can then call the peer.getProvider() method in Exhibit A in order to get a JTAPI provider instance for the specified service.

6) The following table shows the correspondence between the elements of the method claims in the present patent application and elements of the software code in Exhibit A, which includes the following class listings:

- CoreConnectTask.java
- GenCall.java
- GenJtapiPeer.java
- JtsmiPlugin.java
- MultyPluginJTSMI.java
- ProviderPlugin.java
- HybridProviderPlugin.java

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

These classes represent only a portion of the software code required to run GenJTAPI applications, but we believe they illustrate sufficiently our reduction to practice of the elements of the claims. The remaining classes are omitted for brevity.

Claim 1	Exhibit A
1. A method for communication, comprising:	"Telephony" is the form of communication in question.
receiving a request from a first party, submitted via a first communication service provider to a telephony application, to place a call using the application to a second party;	This function is carried out by any sort telephony application, which communicates with GenJTAPI via the JTAPI API. The address (in the form of the string of dialedDigits) of the second party is received and processed by the method connect() listed on page 4 of the GenCall class in Exhibit A.

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

<p>responsive to a characteristic of the call placed by the first party, selecting a second communication service provider to carry the call between the application and the second party; and</p>	<p>The class pluginClass on page 2 of the MultyPluginJTSMI class in Exhibit A associates service providers with address substrings. (The address of the second party is a "characteristic" of the call.) The method createEndpoints(), which is called by init() on page 3 of the MultyPluginJTSMI class, creates a hash table associating service providers with the corresponding plug-ins. These elements are used in providing the pluginForAddress string to GenJTAPI via the JtsmiPlugin interface class in Exhibit A.</p>
<p>connecting the second party via the second communication service provider to communicate with the first party using the application,</p>	<p>The connection is established by the run() method on page 2 of the CoreCallTask class in Exhibit A. The connection uses the appropriate provider plug-ins for the first and second parties, as identified respectively by the connectLocalParty() and connectRemoteParty() methods.</p>

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

wherein receiving the request comprises submitting the request to the application via an application programming interface (API), which exposes a platform-independent call model to the application, and wherein connecting the second party comprises connecting the call responsive to an instruction submitted by the application to the API, and	The JTAPI API of GenJTAPI receives the request from the application. The elements of the platform-independent call model are listed in the comments on page 1 of the CoreCallTask class in Exhibit A. In response to the call instruction submitted by the application to JTAPI, run() calls the appropriate methods of the GenJtapiPeer class in Exhibit A to create the appropriate provider instances (as explained above in paragraph 4). The connection is established using the methods of the GenCall class in Exhibit A (<i>inter alia</i>), particularly the connect() method on page 4 of GenCall.
---	--

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

<p>wherein the first and second communication service providers have respective first and second telephony signaling stacks, and wherein the call model comprises an abstract call model that is independent of the telephony signaling stacks used in placing calls to and receiving calls from the application.</p>	<p>The ProviderPlugin and HybridProviderPlugin classes defined in Exhibit A are used to mediate between the GenJTAPI layer and the telephony stacks of the selected providers. The methods in these classes communicate with GenJTAPI via JTSPi. The GenCall class of GenJTAPI, as noted above, provides the methods of an abstract call model that is independent of the provider-specific plug-ins. (See notes on page 1 of the ProviderPlugin class.)</p>
---	--

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Claim 4	
<p>4. A method according to claim 1, wherein receiving the request comprises passing the request from the first telephony signaling stack to the abstract call model via a service provider interface of the call model, and wherein connecting the second party comprises passing signals to the second telephony signaling stack via the service provider interface, wherein the service provider interface is independent of the telephony signaling stacks.</p>	<p>The methods in the ProviderPlugin class in Exhibit A, such as the prvConnectLocal() and prvConnectRemote() methods, are used to pass signals between the GenJTAPI layer and the respective signaling stacks of the first and second service providers in order to connect a hybrid call. These methods use the JTSPI service provider interface, which is the same for all plug-ins and is thus independent of the telephony signaling stacks.</p>

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Claim 5	
<p>5. A method according to claim 4, wherein passing the request from the first telephony signaling stack comprises using a first plug-in program to associate the signals in the first telephony signaling stack with corresponding elements of the service provider interface, and wherein passing the signals to the second telephony signaling stack comprises using a second plug-in program to associate the signals in the second telephony signaling stack with the corresponding elements of the service provider interface.</p>	<p>The provider plug-in programs are defined by the ProviderPlugin and HybridProviderPlugin classes, as noted above. Each plug-in program includes methods such as ring, answer, disconnect, etc., which handle the corresponding signals in the service provider's signaling stack.</p>

Claim 6	
<p>6. A method according to claim 5, wherein selecting the second communication service provider comprises selecting the second plug-in program from among a plurality of the plug-in programs that are provided for interacting with the abstract call model.</p>	<p>As noted above, the method <code>init()</code> in the <code>MultyPluginJTSMI</code> class creates a table of available plug-ins. The <code>getProviderPlugin()</code> method selects the appropriate second plug-in program based on the address of the second party.</p>
Claim 7	
<p>7. A method according to claim 6, wherein selecting the second plug-in program comprises passing information regarding the call to a service manager program via a service management interface of the abstract call model, wherein the service manager program processes the information to determine the characteristic, and selects the second plug-in program responsive to the characteristic from a registry of the plug-in programs.</p>	<p>The JTSMI service management interface is defined by the <code>MultyPluginJTSMI</code> and <code>JtsmiPlugin</code> classes. The <code>GenJTAPI</code> layer passes information regarding the call, i.e., a string representing the address of the second party, to the JTSMI plug-in via the <code>addressConvert</code> interface, and receives the name of the selected plug-in program via the <code>pluginForAddress</code> interface. The registry of plug-ins is created by the <code>init()</code> method, as explained above.</p>

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Claim 8	
8. A method according to claim 1, wherein receiving the request comprises receiving an address of the second party to whom the call is to be placed, and wherein selecting the second communication service provider comprises parsing the address to determine the second communication service provider that should be selected.	As noted above, the getAddress() method in the MultyPluginJTSMI class receives the address of the party to whom the call is to be placed. The JTSMI plug-in parses the address and then returns the pluginForAddress string to GenJTAPI via the JtsmiPlugin interface to indicate the selection of the second communication service provider.
Claim 9	
9. A method according to claim 8, wherein receiving the address comprises receiving a telephone number, and wherein parsing the address comprises identifying the second communication provider based on a portion of the telephone number.	As explained above, the connect() method on page 4 of the GenCall class receives the "dialedDigits" indicating the telephone number of the second party in the call. The JTSMI plug-in parses this telephone number to identify the second communication provider.

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Claim 10	
<p>10. A method according to claim 1, wherein selecting the second communication service provider comprises determining a communication protocol to be used in communicating with the second party, and choosing the second communication service provider such that the second communication service provider supports the communication protocol.</p>	<p>The GenJTAPI software code provided in Exhibit A was meant to support "hybrid calls," i.e., calls between different service providers with different telephony signaling stack (and thus different protocols). The code itself does not specify or limit the protocols that are to be supported. Rather, the JTSMI plug-in would have parsed the address of the second party in order to identify the appropriate service communication providers, with the appropriate communication protocol and provider plug-in. In regard to this claim, the Examiner indicated that Smyk (U.S. Patent 6,597,686, col. 3, lines 33-35, and col. 6, lines 43-46) would have led a person of ordinary skill in the art to determine the communication protocol and communication service provider in the context of claim 10.</p>

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

	<p>If this rationale were conceded to be correct, then it would have been obvious to the person of ordinary skill to create a JTSMI plug-in with this capability, thus implementing the method of claim 10 on the basis of the GenJTAPI software in Exhibit A.</p>
Claim 11	
<p>11. A method according to claim 10, wherein receiving the request from the first party comprises communicating with the first party via the first communication service provider using a first communication protocol, and wherein the communication protocol used in communicating with the second party comprises a second communication protocol, different from the first protocol.</p>	<p>As noted above in reference to claim 10, the GenJTAPI software code provided in Exhibit A was meant to support "hybrid calls," i.e., calls between different service providers with different telephony signaling stacks (and thus different protocols). The code itself does not specify or limit the protocols that are to be supported but clearly supports call scenarios in which the first and second parties use different protocols. In regard to this claim, the Examiner indicated that Smyk (same</p>

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

	<p>passages as for claim 10) would have led a person of ordinary skill in the art to use different, first and second communication protocols in the context of claim 11. If this rationale were conceded to be correct, then it would have been obvious to the person of ordinary skill to use the GenJTAPI software in this manner, thus implementing the method of claim 11 on the basis of the GenJTAPI software in Exhibit A.</p>
--	---

Claim 12	
<p>12. A method according to claim 11, wherein one of the first and second communication protocols comprises a circuit-switched network protocol, while the other of the first and second communication protocols comprises a packet-switched network protocol.</p>	<p>As noted above in reference to claims 10 and 11, the GenJTAPI software code does not specify or limit the protocols that were to be supported in hybrid calls, but is clearly capable of supporting calls between packet- and circuit-switched networks. In regard to this claim, the Examiner indicated that Smyk (col. 3, lines 33-35) would have led a person of ordinary skill in the art to connect users of circuit-switched and packet-switched network protocols in the context of claim 12. If this rationale were conceded to be correct, then it would have been obvious to the person of ordinary skill to use the GenJTAPI software in this manner, thus implementing the method of claim 12 on the basis of the GenJTAPI software in Exhibit A.</p>

Claim 13	
<p>13. A method according to claim 1, wherein selecting the second communication service provider comprises specifying a selection rule, and applying the selection rule to the characteristic in order to determine the second communication service provider to be selected.</p>	<p>As noted above, the GenJTAPI software code in Exhibit A permits the JTSMI plug-in to implement any sort of rules for selection of the second communication service provider. Although Exhibit A does not explicitly show any specific selection rules, the Examiner indicated that Smyk (col. 5, lines 50-55, and col. 5, line 67 - col. 6, line 3) would have led a person of ordinary skill in the art to specify and apply a selection rule in order to determine the second communication service provider in the context of claim 13. If this rationale were conceded to be correct, then it would have been obvious to the person of ordinary skill to create a JTSMI plug-in with this capability, thus implementing the method of claim 13 on the basis of the GenJTAPI software in Exhibit A.</p>

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Claim 14	
<p>14. A method according to claim 13, wherein specifying the selection rule comprises specifying a temporal criterion, so that the second communication service provider is selected depending on a point in time at which the call is placed.</p>	<p>Although the GenJTAPI software code does not relate to the types of rules that may be implemented by the JTSMI plug-in, the Examiner indicated that Hetz (U.S. Patent 6,185,289, col. 7, lines 52-56), in combination with Smyk, would have led a person of ordinary skill in the art to specify time criteria according to which service providers should be selected in the context of claim 14. If this rationale were conceded to be correct, then it would have been obvious to the person of ordinary skill to create a JTSMI plug-in with this capability, thus implementing the method of claim 14 on the basis of the GenJTAPI software in Exhibit A.</p>

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Claim 15	
<p>15. A method according to claim 1, wherein the telephony application comprises a teleconferencing application, and wherein connecting the second party comprises establishing a teleconference between the first and second parties.</p>	<p>Although the GenJTAPI software code provided in Exhibit A was meant to serve a variety of different telephony applications, the code itself does not specify or limit the types of applications for which it can be used. The Examiner indicated, however, that Smyk (col. 6, lines 30-32 and 43-46) would have led a person of ordinary skill in the art to establish a teleconference between the first and second parties in the context of claim 15. If this rationale were conceded to be correct, then it would have been obvious to the person of ordinary skill to interface a teleconferencing application with the GenJTAPI software so as to implement the method of claim 15 on the basis of the GenJTAPI software in Exhibit A.</p>

Claim 16	
<p>16. A method according to claim 1, wherein the telephony application comprises a call center application, and wherein connecting the second party comprises establishing voice communications between a customer and a call center agent.</p>	<p>As explained above, the GenJTAPI software code provided in Exhibit A does not specify or limit the types of applications for which it can be used. The Examiner indicated, however, that Smyk (col. 6, lines 30-32 and 43-46) would have led a person of ordinary skill in the art to establish voice communications between a customer and a call center agent in the context of claim 16. If this rationale were conceded to be correct, then it would have been obvious to the person of ordinary skill to interface a call center application with the GenJTAPI software so as to implement the method of claim 16 on the basis of the GenJTAPI software in Exhibit A.</p>

7) Claims 26, 29-41, 56 and 59-71 recite apparatus and computer software products, with limitations similar to those of method claims 1 and 4-16. Based on the similarity of subject matter between the method, apparatus and software claims, it can similarly be demonstrated that we reduced to practice the entire invention recited in claims 26, 29-41, 56

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

and 59-71 prior to September 6, 2006.

8) The GenJTAPI software described above was tested in handling actual telephone traffic at the facilities of Sonera (a Finnish telecommunication service provider) prior to September 6, 2006. The successful test is described in an e-mail letter written by the project leader, Phina Vortman, to IBM colleagues shortly after the test. This letter is attached hereto as Exhibit C. The date blacked out of the communication is prior to September 6, 2000. As explained in the letter, GenJTAPI was proven to work for its intended purpose in conjunction with JTSPi and Service Management (JTSMi) in an actual telephony application on real switching equipment provided by Sonera. (We note, incidentally, that GenJTAPI was neither publicly disclosed nor offered for sale in the U.S. prior to January 25, 2001.)

We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and conjecture are thought to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Samuel Kallner
Citizen of Israel
52 Tal Menashe
D.N. Menashe 37867
Israel

Date:

Lev Kozakov
Citizen of Israel
80/8 Hatishbi Street
Haifa 34523
Israel

Date:

Alexey Roytman
Citizen of Israel
68/6 Ben-Zvi Street
Kiryat Ata 28065
Israel

Date:

Uri Shani
Citizen of Israel
71 Givat Adi
17940
Israel

Date:

Pnina Vortman
Citizen of Israel
21 Netiv Ofakim
Haifa 34467
Israel

Date:

US 10/053,872

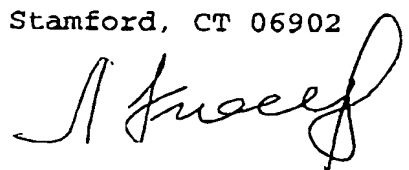
Declaration under 37 C.F.R 1.131 by Kallner et al.

Samuel Kallner
Citizen of Israel
52 Tal Menashe
D.N. Menashe 37867
Israel

Date:

Lev Kozakov
Citizen of Israel
1 Strawberry Hill Avenue,
apt.7G, Stamford, CT 06902
U.S.A.

Date:


Apr 24, 2006

Alexey Roytman
Citizen of Israel
68/6 Ben-Zvi Street
Kiryat Ata 28065
Israel

Date:

Uri Shani
Citizen of Israel
71 Givat Adi
17940
Israel

Date:

Pnina Vortman
Citizen of Israel
21 Netiv Ofakim
Haifa 34467
Israel

Date:

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Samuel Kallner
Citizen of Israel
52 Tal Menashe
D.N. Menashe 37867
Israel

Date:

Samuel Kallner

26/4/2006

Alexey Roytman
Citizen of Israel
68/6 Ben-Zvi Street
Kiryat Ata 28065
Israel

Date:

Pinna Vortman
Citizen of Israel
21 Netiv Ofakim
Haifa 34467
Israel

Date:

Dev Kozakov
Citizen of Israel
60/8 Hatishbi Street
Haifa 34523
Israel

Date:

Uri Shani
Citizen of Israel
71 Givat Adi
17940
Israel

Date:

US 10/053,872

Declaration under 37 C.F.R 1.131 by Kallner et al.

Samuel Kallner
Citizen of Israel
52 Tal Menashe
D.N. Menashe 37867
Israel

Date:

Alexey Roytman
Citizen of Israel
68/6 Ben-Zvi Street
Kiryat Ata 28065
Israel

Date:

25/4/2006

Pnina Vortman

Pnina Vortman
Citizen of Israel
21 Netiv Ofakim
Haifa 34467
Israel

Date:

24.4.06

Lev Kozakov
Citizen of Israel
80/8 Hatishbi Street
Haifa 34523
Israel

Date:

Uri Shani
Citizen of Israel
71 Givat Adi
17940
Israel

Date:

24/4/2006

ANNEX A

```
package com.ibm.hrl.jtapi;

/*
 * (c) Copyright IBM Corporation 1998,1999
 * IBM Research Laboratory in Haifa
 * Generic JTAPI Implementation (JTAPI 1.2)
 * -----
 * Package   : com.ibm.hrl.jtapi
 * Class     : CoreCallTask
 * Created    : ██████████
 */
import java.util.Vector;

import javax.telephony.*;
import javax.telephony.events.Ev;
import javax.telephony.events.CallEv;

import com.ibm.hrl.jtapi.jtspi.ProviderPlugin;
import com.ibm.hrl.jtapi.util.JtapiObjectCreator;

/**
 * Implements ConnectTask for the Core package.
 * Allows asynchronously placing outgoing calls.
 * According to explanation of <code>javax.telephony.Call.connect(), the task
 * does following:
 * <OL>
 * <LI>The originating Connection moves from the <code>Connection.IDLE</code>
 * state into the <code>Connection.CONNECTED</code> state.
 * A <code>TerminalConnection</code> is created in the
 * <code>TerminalConnection.IDLE</code> state and moves to
 * the <code>TerminalConnection.ACTIVE</code> state.
 * <p>
 * <B>Events delivered to the application:</B>
 * a <code>ConnectionEvent.CONNECTION_CONNECTED</code> / <code>ConnConnectedEv</code>
 * for the originating Connection,
 * a <code>TerminalConnectionEvent.TERMINAL_CONNECTION_CREATED</code> /
 * <code>TermConnCreatedEv</code> and a
 * <code>TerminalConnection.TERMINAL_CONNECTION_ACTIVE</code> /
 * <code>TermConnActiveEv</code> for the new <code>TerminalConnection</code>.
 * <p>
 * If the originating <code>Address</code> has more than one <code>Terminal</code>,
 * these additional <code>Terminal</code>s are involved in the telephone call.
 * Additional <code>TerminalConnection</code> objects associated
 * with the originating <code>Connection</code> and these <code>Terminal</code>s
 * are created. For each <code>TerminalConnection</code> created a
 * <code>TerminalConnectionEvent.TERMINAL_CONNECTION_CREATED</code> /
 * <code>TermConnCreatedEv</code> is delivered. These <code>TerminalConnection</code>s
 * will be in the <code>TerminalConnection.PASSIVE</code> state and a
 * <code>TerminalConnectionEvent.TERMINAL_CONNECTION_PASSIVE</code> /
 * <code>TermConnPassiveEv</code> is delivered for each.
 * <p>
 * <LI>The destination <code>Connection</code> moves into the
 * <code>Connection.INPROGRESS</code> state as the Call proceeds.
 * <p>
 * <B>Events delivered to the application:</B>
 * a
 * <code>ConnectionEvent.CONNECTION_IN_PROGRESS</code> / <code>ConnInProgressEv</code>
 * for the destination <code>Connection</code>.
```

```

* <p>
* <LI>Next steps of the connection sequence operation are done as ringing callback.
* </OL>
*
* @see javax.telephony.Call#connect(Terminal, Address, String)
*
* @author Alexey Roytman
*/

```

```

public class CoreConnectTask implements Runnable
{

```

```

    // IBM Copyright
    public static final String IBM_Copyright = Copyright.SHORT_STRING;

```

```

    /*
     * Attributes
     */

```

```

    protected GenCall m_call;

```

```

    protected GenTerminal m_localTerminal;

```

```

    protected GenAddress m_localAddress, m_destAddress;

```

```

    protected int m_observCause;

```

```

    protected int m_observMeta;

```

```

    protected int m_opCode;

```

```

    // ctors .....

```

```

    /**

```

```

     * Constructor for outgoing call task.

```

```

     *

```

```

     * @param call The given outgoing Call.

```

```

     * @param localAddress The given local <core>Address</code>

```

```

     * @param localTerminal The given local <core>Terminal</core>

```

```

     * @param destAddress The given destination <core>Address</code>

```

```

     */

```

```

    public CoreConnectTask( GenCall call, GenAddress localAddress,
                           GenTerminal localTerminal, GenAddress destAddress )

```

```

    {

```

```

        m_call = call;

```

```

        m_localAddress = localAddress;

```

```

        m_localTerminal = localTerminal;

```

```

        m_destAddress = destAddress;

```

```

        m_opCode = CoreOperations.CALL_CONNECT_OP;

```

```

        m_observCause = Ev.CAUSE_NORMAL;

```

```

        m_observMeta = Ev.META_CALL_STARTING;

```

```

    }

```

```

    public void run()

```

```

    {

```

```

        String callId = m_call.getId();

```

```

        String localAddressName = m_localAddress.getName();

```

```

        String destAddressName = m_destAddress.getName();

```

```

        synchronized( callId )

```

```

        {

```

```

GenConnection localConnection =
    m_call.getConnection(localAddressName );
if( null == localConnection )
{
    GenJtapiPeer.ms_log.errorMessage(
        "CoreConnectTask: cannot get local connection");
    return;
}
GenConnection remoteConnection =
    m_call.getConnection(destAddressName);
if( null == remoteConnection )
{
    GenJtapiPeer.ms_log.errorMessage(
        "CoreConnectTask: cannot get remote connection");
    return;
}
if( connectLocalParty( localConnection ) )
{
    connectRemoteParty( remoteConnection );
}
}
}

```

```

protected boolean connectLocalParty( GenConnection connection )
{
    ProviderPlugin provPlugin = m_localAddress.getProviderPlugin();
    GenProvider provider = (GenProvider)m_call.getProvider();
    JtapiObjectCreator creator = provider.getObjectCreator();
    // address, we should create PASSIVE terminal connection
    GenTerminal[] terminals = (GenTerminal[])m_localAddress.getTerminals();
    String localAddressName = m_localAddress.getName();
    String callId = m_call.getId();
    for(int i=0; i<terminals.length; i++)
    {
        try
        {
            provPlugin.prvConnectLocal(callId, localAddressName,
                terminals[i].getName());
        }
        catch ( Exception exc )
        {
            GenJtapiPeer.ms_log.errorMessage(
                "CoreConnectTask: placeOutgoingCall: ", exc);
            m_call.disconnectPart( localAddressName, terminals[i].getName(),
                0, CoreOperations.FAILED_CB);
            return false;
        }
        GenTerminalConnection tconn =
            creator.createTerminalConnection(terminals[i], connection,
                m_observCause, m_observMeta,
false);

        if(terminals[i] != m_localTerminal)
        {
            tconn.moveToState( TerminalConnection.PASSIVE, m_observCause,
                m_observMeta, false, m_opCode);
        }
        else
        {

```

```

        tconn.moveToState(TerminalConnection.ACTIVE, m_observCause,
                           m_observMeta, false, m_opCode);
    }
}

// move local. connection to CONNECTED state
connection.moveToState( Connection.CONNECTED, m_observCause,
                        m_observMeta, false, m_opCode);
return true;
}

/**
 * This method serves as the final part of a standard routine for
 * connection of outgoing call.
 * <p>
 * Moves destination (remote) connection to the INPROGRESS state.
 *
 * @param connection The given destination <code>Connection</code>.
 * @return <code>true</code> if the method is succeed, <code>false</code>
 * otherwise.
 */
protected boolean connectRemoteParty ( GenConnection connection )
{
    // get required objects
    ProviderPlugin provPlugin = m_destAddress.getProviderPlugin();
    String callId = m_call.getId();
    String destAddressName = m_destAddress.getName();
    GenJtapiPeer.ms_log.progressMessage(
        "CoreConnectTask: Connecting remote party: " + destAddressName);
    try
    {
        provPlugin.prvConnectRemote ( callId, destAddressName);
    }
    catch ( Exception exc )
    {
        GenJtapiPeer.ms_log.errorMessage(
            "CoreConnectTask: Connecting remote party: ", exc);
        try
        {
            // roolback
            connection.disconnect();
        }
        catch(Exception e)
        {
            GenJtapiPeer.ms_log.warningMessage(
                "CoreConnectTask: Connecting remote party: ", e);
        }
        return false;
    }
    // move dest connection to INPROGRESS state
    connection.moveToState( Connection.INPROGRESS, m_observCause,
                            m_observMeta, false, m_opCode);
    return true;
}
}

```

```

package com.ibm.hrl.jtapi;

/*
 * (c) Copyright IBM Corporation 1998,1999,2000
 * IBM Research Laboratory in Haifa
 * Generic JTAPI Implementation (JTAPI 1.3)
 * -----
 * Package      : com.ibm.hrl.jtapi
 * Class        : GenCall
 * Created      : ██████████
 */

import java.util.Vector;

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.capabilities.CallCapabilities;

import com.ibm.hrl.jtapi.capabilities.*;
import com.ibm.hrl.jtapi.events.*;
import com.ibm.hrl.jtapi.jtspi.ProviderPlugin;
import com.ibm.hrl.jtapi.jtspi.JtspiException;
import com.ibm.hrl.jtapi.util.JtapiObjectCreator;
import com.ibm.hrl.jtapi.util.JtapiObjectNotifier;
import com.ibm.hrl.util.ArrayUtils;

import com.ibm.telephony.ics.ICSCall;

/**
 * Generic implementation of the <code>Call</code> interface.
 * for the Core package.
 *
 * @see javax.telephony.Call
 *
 * @author Lev Kozakov
 * @author Alexey Roytman
 */
public class GenCall implements Call, ICSCall
{
    // IBM Copyright
    public static final String IBM_Copyright = Copyright.SHORT_STRING;

    protected Vector m_connections;
    protected GenAddress m_origAddress;
    protected GenTerminal m_origTerminal;
    protected GenAddress m_destAddress;
    protected Vector m_observers;
    // Added support for the Listener model of JTAPI 1.3
    protected Vector m_listeners;
    protected GenProvider m_provider;
    protected boolean m_isFirstParty;
    protected int m_coreState = Call.IDLE;
    protected String m_id;
    protected GenCallCapabilities m_statCapabilities;

    // ctors .....
    /**
     * Constructs new <code>GenCall</code> object, using reference to a given
     * <code>GenProvider</code>, a ID of the created <code>Call</code> and flag,

```

```

* which define party of the <code>GenCall</code>.
*
* @param provider      The reference to <code>GenProvider</code>
* @param id            The identifier (name) of the ,code>GenCall</code>
* @param isFirstParty  <code>true</code> if the call is first party call,
*                      <code>false</code> otherwise.
*/
public GenCall(GenProvider provider, String id, boolean isFirstParty )
    throws InvalidStateException
{
    provider.assertIN_SERVICE(
        "Provider should be in the \"IN_SERVICE\" state to create new call");
    m_id = id;
    m_isFirstParty = isFirstParty;
    m_provider = provider;
    m_connections = new Vector();
    m_observers   = new Vector();
    m_listeners   = new Vector();
    m_statCapabilities = (GenCallCapabilities)provider.getCallCapabilities();
}

// JTAPI methods .....

/**
 * Adds an listener to this <code>Call</code> object.
 * <p>
 * This method implements the javax.telephony.Call.addCallListener(CallListener)
 * method.
 * <p>
 * <B>Post-conditions:</B>
 * <OL>
 * <LI>listener is an element of this.getCallListeners().
 * <LI>A snapshot of events is delivered to the listener, if appropriate.
 * </OL>
 *
 * @param listener The listener being added.
 *
 * @exception javax.telephony.MethodNotSupportedException if the observer
 * cannot be added at this time.
 * @exception javax.telephony.ResourceUnavailableException if the resource
 * limit for the number of listeners has been exceeded.
 *
 * @see javax.telephony.Call#addCallListener
 */
public void addCallListener(CallListener listener)
    throws ResourceUnavailableException, MethodNotSupportedException
{
    synchronized(m_listeners)
    {
        if ( listener == null || m_listeners.contains(listener) )
        {
            return;
        }
        m_listeners.addElement(listener);
    }
    notifyStateSnapshot(listener);
}

/**

```

```

* Returns an array of <code>Connection</code> objects associated with this
* <code>GenCall</code>.
* <p>
* This method implements the
* <code>javax.telephony.Call.getConnections()</code> method.
* <p>
* <B>Post-conditions:</B>
* <OL>
* <LI>Let Connection[] conn = Call.getConnections()
* <LI>if this.getState() == Call.IDLE then conn = null
* <LI>if this.getState() == Call.INVALID then conn = null
* <LI>if this.getState() == Call.ACTIVE then conn.length >= 1
* <LI>For all i, conn[i].getState() != Connection.DISCONNECTED
* </OL>
*
* @return An array of <code>Connection</code> objects associated with this
* <code>Call</code>.
*
* @see javax.telephony.Call#getConnections
*/
public Connection[] getConnections()
{
    synchronized(m_connections)
    {
        if( m_connections.size () == 0 )
        {
            return null;
        }
        GenConnection ret[] = new GenConnection[m_connections.size()];
        m_connections.copyInto( ret );
        return ret;
    }
}

/**
* Places a telephone call from an originating endpoint to a destination
* address string.
* <p>
* This method implements the <code>javax.telephony.Call.connect()</code>
* method.
* <p>
* <B>Pre-conditions:</B>
* <OL>
* <LI>(this.getProvider()).getState() == Provider.IN_SERVICE
* <LI>this.getState() == Call.IDLE
* </OL>
* <B>Post-conditions:</B>
* <OL>
* <LI>(this.getProvider()).getState() == Provider.IN_SERVICE
* <LI>this.getState() == Call.ACTIVE
* <LI>Let Connection c[] = this.getConnections()
* <LI>c.length == 2
* <LI>c[0].getState() == Connection.IDLE (at least)
* <LI>c[1].getState() == Connection.IDLE (at least)
* </OL>
*
* @param origterm The given originating <code>Terminal</code>.
* @param origaddr The given originating <code>Address</code>.
* @param destString The given destination endpoint string.

```

```

*
* @exception javax.telephony.ResourceUnavailableException if an internal
* resource necessary for placing the phone call is unavailable.
* @exception javax.telephony.PrivilegeViolationException if the
* application does not have the proper authority to place a telephone
* call.
* @exception javax.telephony.InvalidPartyException if either the
* originator or the destination does not represent a valid party
* required to place a telephone call.
* @exception javax.telephony.InvalidArgumentException if an argument
* provided is not valid either by not providing enough information for
* this method or is inconsistent with another argument.
* @exception javax.telephony.InvalidStateException if some object
* required by this method is not in a valid state as designated by the
* pre-conditions for this method.
* @exception javax.telephony.MethodNotSupportedException if the
* implementation does not support this method.
*
* @see javax.telephony.Call#connect
* @see com.ibm.hrl.jtapi.CoreConnectTask
*/
public Connection[] connect(Terminal localTerm, Address localAddr,
                           String dialedDigits)
    throws ResourceUnavailableException,
           PrivilegeViolationException, InvalidPartyException,
           InvalidArgumentException, InvalidStateException,
           MethodNotSupportedException
{
    // TBD: check application privileges
    // check capabilities
    if( ! m_statCapabilities.canConnect() )
    {
        throw new MethodNotSupportedException(
            " GenCall: connect: method is not supported");
    }
    // check arguments
    GenTerminal.assertValid( localTerm );
    GenAddress.assertValid( localAddr );
    if(null == dialedDigits)
    {
        throw new InvalidArgumentException("distination address is null");
    }

    // check preconditions for main objects
    m_provider.assertIN_SERVICE(
        "Provider should be in the \"IN_SERVICE\" state for Call.connect operation");

    GenTerminal localTerminal = (GenTerminal)localTerm;
    GenAddress localAddress = (GenAddress)localAddr;

    // localTerm and localAddr should be local
    localTerminal.assertLocal();
    localAddress.assertLocal();

    // there should be association between origterm and origaddr
    if(!localTerminal.isAssociated( localAddress ))
    {
        throw new InvalidPartyException(

```



```

        InvalidPartyException.ORIGINATING_PARTY,
        "GenCall.connect: no address-terminal association" );
    }
    m_origAddress = localAddress;
    m_origTerminal = localTerminal;

    GenConnection[] newConnections = new GenConnection[2];
    JtapiObjectCreator creator = m_provider.getObjectCreator();

    synchronized(m_id)
    {
        assertState(
            "The Call should be in the \"IDLE\" state for Call.connect operation",
            Call.IDLE );
        try
        {
            addObservers(localAddress.getCallObservers());
            addCallListeners(localAddress.getCallListeners());
            Terminal[] terminals = localAddress.getTerminals();
            for(int i=0; i< terminals.length; i++)
            {
                addObservers(terminals[i].getCallObservers());
                addCallListeners(terminals[i].getCallListeners());
            }
        }
        catch( Exception e )
        {
            /* The "possible" exceptions are ResourceUnavailableException and
               MethodNotSupportedException. These exception cannot be thrown
               in our implementation in this place. */
            GenJtapiPeer.ms_log.warningMessage("GenCall: connect " + e);
        }
    }

    /*
    1. The Call.connect() method is invoked with the given arguments.
       Two Connection objects are created and returned, each in the
       Connection.IDLE state.

       Events delivered to the application: a CallActivEv and
       two ConnCreatedEv, one for each Connection.
    */
    int observMeta = Ev.META_CALL_STARTING, observCause = Ev.CAUSE_NORMAL;
    ProviderPlugin provPlugin = localAddress.getProviderPlugin();

    if( m_provider.isLocalAddress( dialedDigits ))
    {
        m_destAddress = (GenAddress)m_provider.getAddress( dialedDigits );
    }
    else
    {
        m_destAddress = creator.createRemoteAddress(m_provider, provPlugin,
                                                    dialedDigits);
    }

    newConnections[ 0 ] = creator.createConnection( this, localAddress,
                                                    observCause, observMeta, true );
    newConnections[ 1 ] = creator.createConnection(this, m_destAddress,
                                                    observCause, observMeta, false);

    moveToState( Call.ACTIVE, observCause, observMeta, false,

```

```

        CoreOperations.CALL_CONNECT_OP);
    // start placing call
    m_provider.runIt( new CoreConnectTask( this, localAddress,
        localTerminal, m_destAddress ));
}
return newConnections;
}

/**
 * Adds a given <code>CallObserver</code> object to this <code>Call</code>.
 * <p>
 * This method implements the <code>javax.telephony.Call.addObserver()
 * </code> method.
 * <p>
 * <B>Post-Conditions:</B>
 * <OL>
 * <LI>observer is an element of <code>this.getObservers()</code>
 * <LI>A snapshot of events is delivered to the observer, if appropriate.
 * </OL>
 *
 * @param observer The given <code>CallObserver</code> to be added.
 *
 * @exception javax.telephony.ResourceUnavailableException if the resource
 * limit for the number of observers has been exceeded.
 * @exception javax.telephony.MethodNotSupportedException if the observer
 * cannot be added at this time
 *
 * @see javax.telephony.Call#addObserver
 */
public void addObserver(CallObserver observer)
    throws ResourceUnavailableException, MethodNotSupportedException
{
    if( ! m_statCapabilities.isObservable() )
    {
        throw new MethodNotSupportedException(
            " GenCall: addObserver: method is not supported");
    }
    synchronized(m_observers)
    {
        if(null == observer || m_observers.contains(observer))
        {
            return;
        }
        m_observers.addElement(observer);
    }
    notifyStateSnapshot( observer );
}

/**
 * Returns a list of all <code>CallListeners</code> objects associated
 * with this <code>Call</code> object.
 * <p>
 * This method implements the <code>javax.telephony.Call.getCallListeners()
 * </code> method.
 * <p>
 * <B>Post-conditions:</B>
 * <OL>
 * <LI>Let CallListeners[] listeners = this.getCallListeners()
 * <LI>listeners == null or listeners.length >= 1

```

```

* </OL>
*
* @return An array of all <code>CallListeners</code> objects associated
* with this <code>Call</code>.
*
* @see javax.telephony.Call#getCallListeners
*/
public CallListener[] getCallListeners()
{
    synchronized ( m_listeners )
    {
        if (m_listeners.size() == 0)
        {
            return null;
        }
        CallListener[] array = new CallListener[m_listeners.size()];
        m_listeners.copyInto(array);
        return array;
    }
}

/**
* Returns an array of all <code>CallObserver</code> objects on
* this <code>Call</code>.
* <p>
* This method implements the <code>javax.telephony.Call.getObservers()
* </code> method.
* <p>
* <B>Post-Conditions:</B>
* <OL>
* <LI>Let CallObserver[] obs = this.getObservers()
* <LI>obs == null or obs.length >= 1
* </OL>
*
* @return An array of all <code>CallObserver</code> objects on this
* <code>Call</code>.
*/
public CallObserver[] getObservers()
{
    synchronized( m_observers )
    {
        if( m_observers.size() == 0 )
        {
            return null;
        }
        CallObserver[] array = new CallObserver[m_observers.size()];
        m_observers.copyInto( array );
        return array;
    }
}

/**
* Removes the given observer from this <code>GenCall</code>.
* <p>
* This method implements the <code>javax.telephony.Call.removeObserver()
* </code> method.
* <p>
* <B>Post-Conditions:</B>
* <OL>

```

```

* <LI>observer is not an element of this.getObservers()
* <LI>CallObservationEndedEv is delivered to the application
* </OL>
*
* @param observer The <code>CallObserver</code> to be removed.
*
* @see javax.telephony.Call#removeObserver
* @see com.ibm.hrl.jtapi.GenCallObservationEndedEv
*/
public void removeObserver(CallObserver observer)
{
    if( null != observer )
    {
        if( m_observers.removeElement( observer ))
        {
            notifyObserverRemoved( observer );
        }
    }
}

/**
* Returns the <code>Provider</code> associated with this <code>GenCall</code>.
* <p>
* This method implements the <code>javax.telephony.Call.getProvider()</code>
* method.
*
* @return The <code>Provider</code> associated with this <code>GenCall</code>.
*/
public final Provider getProvider()
{
    return m_provider;
}

/**
* Returns the dynamic capabilities for the instance of the
* <code>Call</code> object with respect to a <code>Terminal</code>
* and an <code>Address</code>.
* <p>
* This method implements the <code>javax.telephony.Call.getCapabilities()</code>
* method.
*
* @return The dynamic <code>Call</code> capabilities.
*
* @param terminal Dynamic capabilities are with respect to this
* <code>Terminal</code>.
* @param address Dynamic capabilities are with respect to this
* <code>Address</code>.
*
* @exception javax.telephony.InvalidArgumentException indicates the
* <code>Terminal</code> and/or <code>Address</code> argument provided
* was not valid.
*/
public CallCapabilities getCapabilities(Terminal terminal, Address address)
    throws InvalidArgumentException
{
    return GenDynCallCapabilities.createDynCapabilities( m_statCapabilities,
        this, (GenAddress)address, (GenTerminal)terminal);
}

```

```

/**
 * Gets the CallCapabilities object with respect to a
 * Terminal and an Address. If null
 * is passed as a Terminal parameter, the general/provider-wide
 * Call capabilities are returned.
 * <p>
 * This method implements the 
 * javax.telephony.Call.getCallCapabilities(Terminal, Address) method.
 *
 * @deprecated Since JTAPI v1.2. This method has been replaced by the
 * Call.getCapabilities() method. Now the method invokes the
 * Call.getCapabilities() method with the given Terminal
 * and Address arguments.
 *
 * @return The CallCapabilities object with respect to a
 * Terminal and an Address.
 *
 * @param terminal The given Terminal.
 * @param address The given Address.
 *
 * @exception javax.telephony.InvalidArgumentException indicates the
 * Terminal and/or Address argument provided
 * was not valid.
 * @exception javax.telephony.PlatformException if platform-specific
 * exception occurred.
 */
public final CallCapabilities getCallCapabilities(Terminal term, Address addr)
    throws InvalidArgumentException, PlatformException
{
    return getCapabilities(term, addr);
}

/**
 * Returns the current Core state of the telephone Call.
 * <p>
 * This method implements the javax.telephony.Call.getState()
 * method.
 *
 * @return The current state of the telephone Call.
 */
public final int getState()
{
    return m_coreState;
}

/**
 * Removes the given listener from the Call.
 * <p>
 * This method implements the
 * javax.telephony.Call.removeCallListener() method.
 * <p>
 * <B>Post-conditions:</B>
 * <OL>
 * <LI>listener is not an element of this.removeProviderListener()
 * <LI>ProviderEvent with id CALL_EVENT_TRANSMISSION_ENDED
 * is delivered to listener
 * </OL>
 *
 * @param listener The Provider Listener to be removed.

```

```

    *
    * @see javax.telephony.Call#removeCallListener
    * @see com.ibm.hrl.jtapi.GenCall#addCallListener
    */
public void removeCallListener(CallListener listener)
{
    if (listener != null)
    {
        if (m_listeners.removeElement(listener))
        {
            notifyListenerRemoved( listener);
        }
    }
}

// ICS exstantions .....

/**
 * Transfers the given Core state of the telephone <code>Call</code>
 * in the form of a String.
 * <p>
 * This is an extensionn from ZRL.
 *
 * @param state the given state
 * @return The current state of the telephone <code>Call</code> as a String.
 */
public String getStateName(int state)
{
    switch (state)
    {
        case Call.IDLE:
            return "Idle";
        case Call.ACTIVE:
            return "Active";
        case Call.INVALID:
            return "Invalid";
        default:
            return "Unknown";
    }
}

/**
 * Returns the current Core state of the telephone <code>Call</code>.
 * in the form of a String.
 * <p>
 * This is an extension from ZRL.
 *
 * @return The current state of the telephone <code>Call</code> as a String.
 */
public String getStateName()
{
    return getStateName(m_coreState);
}

// Implementation methods .....

/**
 * Returns <code>String</code> representation of the <code>GenCall</code>

```

```

*
* @return <code>String</code> representation of the <code>GenCall</code>
*/
public String toString()
{
    String className = getClass().getName();
    StringBuffer buff = new StringBuffer(
        className.substring( className.lastIndexOf( '.' ) + 1 ) );
    buff.append( " { has " );
    buff.append( m_connections.size() );
    buff.append( " connections } in the state " );
    buff.append( getStateName() );
    return buff.toString();
}

/**
* Returns the global unique ID of this call object.
*
* @return The global unique ID of this call object.
*/
public final String getId()
{
    return m_id;
}

/**
* Returns <code>true</code> if this <code>GenCall</code> is first party
* call, <code>false</code> otherwise.
*
* @return <code>true</code> if this <code>GenCall</code> is first party
* call, <code>false</code> otherwise.
*/
public final boolean isFirstParty()
{
    return m_isFirstParty;
}

/**
* Returns <code>true</code> if this <code>GenCall</code> contains the
* given <code>TerminalConnection</code>, <code>false</code> otherwise.
*
* @return <code>true</code> if this <code>GenCall</code> contains the
* given <code>TerminalConnection</code>, <code>false</code> otherwise.
*
* @exception IllegalArgumentException in the case of null argument
*/
protected boolean contains(TerminalConnection tc)
{
    if( null == tc)
    {
        throw new IllegalArgumentException(
            "GenCall.contains(TerminalConnection): null argument" );
    }
    synchronized( m_connections )
    {
        int size = m_connections.size();
        TerminalConnection[] tConnections;
        for(int i=0; i< size; i++)
        {

```

```

        tConnections = ((GenConnection)m_connections.elementAt(i)).
                        getTerminalConnections();
        for( int j=0; j<tConnections.length; j++)
        {
            if( tConnections[j] == tc)
            {
                return true;
            }
        }
        return false;
    }
}

/**
 * Throws InvalidStateException with given messag, if this
 * <code>GenCall</code> is not in the given state.
 *
 * @param message the message for exception.
 * @param state the given call state.
 * @exception javax.telephony.InvalidStateException Call is not in the
 * given state.
 */
public void assertState(String message, int state) throws InvalidStateException
{
    int callState = m_coreState;
    if ( state != callState ) {
        throw new InvalidStateException( this,
                                        InvalidStateException.CALL_OBJECT,
                                        callState, message );
    }
}

/**
 * Throws InvalidStateException with default messag, if this
 * <code>GenCall</code> is not in the given state.
 *
 * @exception javax.telephony.InvalidStateException Call is not in the
 * given state.
 */
public void assertState(int state) throws InvalidStateException
{
    String message = "the call is not " + getStateName( state );
    assertState( message, state);
}

/**
 * Moves this <code>GenCall</code> to the given state.
 * Notifies registered observers and listeners about the last
 * state change.
 *
 * @param state The given state value.
 * @param cause The given cause ID.
 * @param meta The given meta-code.
 * @param isNewMetaEv <code>true</code> if the change starts new sequence of
 * meta events, <code>false</code> otherwise.
 * @param opCode The given operation code.
 * @exception javax.telephony.InvalidArgumentException The given state is not
 * valid Call state.

```



```

    */
    public void moveToState( int state, int cause, int meta,
                           boolean isNewMetaEv, int opCode)
    {
        synchronized(m_id)
        {
            if( state == m_coreState )
            {
                return;
            }
            switch ( state )
            {
                case Call.INVALID :
                case Call.IDLE :
                case Call.ACTIVE :
                    m_coreState = state;
                    break;
                default :
                    throw new IllegalArgumentException( "invalid Call state - " + state
);
            }
            stateChanged( cause, meta, isNewMetaEv );
        }
    }

    /**
     * Notifies all registered observers and listeners asynchronously
     * about the last state change.
     *
     * @param cause The given cause code.
     * @param meta The given meta-code.
     * @param isNewMetaEv <code>true</code> if the change starts new sequence of
     * meta events, <code>false</code> otherwise.
     */
    protected void stateChanged( int cause, int meta, boolean isNewMetaEv )
    {
        CallObserver[] obList = getObservers();
        if( obList != null )
        {
            CallEv[] evList = getLastStateObEvents( cause, meta, isNewMetaEv);
            if( evList != null )
            {
                JtapiObjectNotifier.getInstance().enqueue( obList, evList );
            }
        }
        CallListener[] lsList = getCallListeners();
        if (lsList != null)
        {
            CallEvent[] evList = getLastStateListenEvents( cause, meta, isNewMetaEv);
            if( evList != null )
            {
                JtapiObjectNotifier.getInstance ().enqueue ( lsList, evList );
            }
        }
    }

    /**
     * Returns array of CallEv objects, associated with the last state change.

```

```

*
* @return The array of CallEv objects, associated with the last
* state change.
* @param cause The given cause code.
* @param meta The given meta-code.
* @param isNewMetaEv <code>true</code> if the change starts new sequence of
* meta events, <code>false</code> otherwise.
*/
protected CallEv[] getLastStateObEvents( int cause, int meta,
                                         boolean isNewMetaEv)
{
    CallEv[] events = new CallEv[1];
    switch ( m_coreState )
    {
        case Call.ACTIVE :
            events[0] = new GenCallActiveEv( this, cause, meta, isNewMetaEv );
            return events;
        case Call.INVALID :
            events[0] = new GenCallInvalidEv( this, cause, meta, isNewMetaEv );
            return events;
        default :
            return null;
    }
}

/**
* Returns array of CallEvent objects, associated with the last state change.
*
* @return The array of CallEvent objects, associated with the last
* state change.
* @param cause The given cause code.
* @param meta The given meta-code.
* @param isNewMetaEv <code>true</code> if the change starts new sequence of
* meta events, <code>false</code> otherwise.
*/
protected CallEvent[] getLastStateListenEvents( int cause, int meta,
                                                boolean isNewMetaEv)
{
    CallEvent[] events = new CallEvent[1];
    switch ( m_coreState )
    {
        case Call.ACTIVE:
        {
            events[0] = new GenCallEvent( this, CallEvent.CALL_ACTIVE,
cause);
            break;
        }
        case Call.INVALID:
        {
            events[0] = new GenCallEvent( this, CallEvent.CALL_INVALID,
cause);
            break;
        }
        default:
        {
            return null;
        }
    }
    return events;
}

```

```

    }

/**
 * Notifies a given call observer asynchronously
 * about the current state snapshot.
 *
 * @param observer The given call observer.
 */
protected void notifyStateSnapshot( CallObserver observer )
{
    CallObserver[] obList = { observer };
    CallEv[] evList =
        getLastStateObEvents( Ev.CAUSE_SNAPSHOT, Ev.META_SNAPSHOT, true);
    if( evList != null )
    {
        GenConnection[] connections = (GenConnection[])getConnections();
        if( null == connections )
        {
            JtapiObjectNotifier.getInstance().enqueue( obList, evList );
            return;
        }
        Vector vector = new Vector();
        ArrayUtils.append( vector, evList );

        for( int i = 0; i < connections.length; i++ )
        {
            CallEv[] connEvList = connections[i].getLastCoreStateEvents(
                Ev.CAUSE_SNAPSHOT, Ev.META_SNAPSHOT, false);
            if( connEvList != null )
            {
                ArrayUtils.append( vector, connEvList );
            }
            TerminalConnection[] tconArray =
                connections[i].getTerminalConnections();
            if( tconArray != null )
            {
                for( int j = 0; j < tconArray.length; j++ )
                {
                    GenTerminalConnection tconItem =
                        (GenTerminalConnection)tconArray [j];
                    CallEv[] tconEvList = tconItem.getLastCoreStateEvents(
                        Ev.CAUSE_SNAPSHOT, Ev.META_SNAPSHOT, false );
                    if( tconEvList != null )
                    {
                        ArrayUtils.append( vector, tconEvList );
                    }
                }
            }
        }
        if( vector.size () > 0 )
        {
            // copy to array
            evList = new CallEv[vector.size ()];
            vector.copyInto( evList );
        }
        JtapiObjectNotifier.getInstance().enqueue( obList, evList );
    }
}

```

```

/**
 * Notifies a given call listener asynchronously
 * about the current state snapshot.
 *
 * @param listener The given call listener.
 */
protected void notifyStateSnapshot(CallListener listener)
{
    CallListener[] lsList = { listener };
    CallEvent[] evList = getLastStateListenEvents(CallEvent.CAUSE_SNAPSHOT, 0,
true);

    if (evList != null)
    {
        GenConnection[] connections = (GenConnection[])getConnections();
        if( null == connections )
        {
            JtapiObjectNotifier.getInstance().enqueue( lsList, evList );
            return;
        }
        Vector vector = new Vector();
        ArrayUtils.append( vector, evList );
        for( int i = 0; i < connections.length; i++ )
        {
            CallEvent[] connEvList =
connections[i].getLastCoreStateEventsEx(CallEvent.CAUSE_SNAPSHOT, 0);
            if( connEvList != null )
            {
                ArrayUtils.append( vector, connEvList );
            }
            TerminalConnection[] tconArray =
connections[i].getTerminalConnections();
            if ( tconArray != null )
            {
                for ( int j = 0; j < tconArray.length; j++ )
                {
                    GenTerminalConnection tconItem =
(GenTerminalConnection)tconArray [j];
                    CallEvent[] tconEvList =
tconItem.getLastCoreStateEventsEx(CallEvent.CAUSE_SNAPSHOT,
0,false);

                    if ( tconEvList != null )
                    {
                        ArrayUtils.append ( vector, tconEvList );
                    }
                }
            }

            if ( vector.size () > 0 )
            {
                // copy to array
                evList = new CallEvent[vector.size ()];
                vector.copyInto( evList );
            }
            JtapiObjectNotifier.getInstance().enqueue( lsList, evList );
        }
    }
}

```

```

}

/**
 * Returns the calling <code>Address</code> associated with this
 * <code>Call</code>.
 *
 * @return The calling <code>Address</code> associated with this
 * <code>Call</code>.
 */
public final Address getCallingAddress()
{
    return m_origAddress;
}

/**
 * Returns the calling <code>Terminal</code> associated with this
 * <code>Call</code>.
 *
 * @return The calling <code>Terminal</code> associated with this
 * <code>Call</code>.
 */
public final Terminal getCallingTerminal()
{
    return m_origTerminal;
}

/**
 * Returns the called <code>Address</code> associated with this
 * <code>Call</code>.
 *
 * @return The called <code>Terminal</code> associated with this
 * <code>Call</code>.
 */
public final Address getCalledAddress()
{
    return m_destAddress;
}

/**
 * Drops the given <code>Call</code>
 *
 * @param opCode the code of operation, which was cause of the drop.
 *
 * @exception javax.telephony.ResourceUnavailableException if an internal
 * resource necessary for the operation is unavailable.
 * @exception javax.telephony.PrivilegeViolationException if the
 * application does not have the proper authority to drop the telephone
 * call.
 * @exception javax.telephony.InvalidStateException if some object
 * required by this method is not in a valid state for this method.
 * @exception javax.telephony.MethodNotSupportedException if the
 * implementation does not support this method.
 */
protected void drop( int opCode)
    throws InvalidStateException, MethodNotSupportedException,
        PrivilegeViolationException, ResourceUnavailableException
{
    int meta = Ev.META_CALL_ENDING, cause = Ev.CAUSE_CALL_CANCELLED;

```

```

synchronized(m_id)
{
    synchronized( m_connections )
    {
        int size = m_connections.size();
        GenConnection conn;
        for(int i=0; i< size; i++)
        {
            conn = (GenConnection)m_connections.elementAt(i);
            conn.disconnect(cause, meta, false, false, opCode);
        }
        m_connections.removeAllElements();
    }
    moveToState(Call.INVALID, cause, meta, true, opCode );
    m_provider.removeCall(this);
}

/**
 * Adds <code>CallObserver</code>s from the given array to this call.
 * If the given array is <code>null</code> this method fails silently,
 * i.e. no observers are added and no exception is thrown.
 *
 * <B>Post-Conditions:</B>
 * <OL>
 * <LI>each observer, from the observers array, is an element of
 * <code>this.getObservers()</code>
 * <LI>A snapshot of events is delivered to each added observer.
 * </OL>
 *
 * @param observers The array of observers being added.
 * @exception MethodNotSupportedException The observer cannot be added at this time
 * @exception ResourceUnavailableException The resource limit for the
 * numbers of observers has been exceeded.
 */
public void addObserver(CallObserver[] observers)
    throws ResourceUnavailableException, MethodNotSupportedException
{
    if(null == observers)
    {
        return;
    }
    for(int i=0; i< observers.length; i++ )
    {
        addObserver(observers[i]);
    }
}

/**
 * Adds <code>CallListener</code>s from the given array to this call.
 * If the given array is <code>null</code> this method fails silently,
 * i.e. no listeners are added and no exception is thrown.
 *
 * <B>Post-Conditions:</B>
 * <OL>
 * <LI>each listener, from the observers array, is an element of
 * <code>this.getCallListeners()</code>
 * <LI>A snapshot of events is delivered to to each added listener.
 * </OL>

```

```

* @param listeners The array of listeners being added.
* @exception MethodNotSupportedException The listener cannot be added at
* this time
* @exception ResourceUnavailableException The resource limit for the
* numbers of listeners has been exceeded.
*/
protected void addCallListeners(CallListener[] listeners)
    throws ResourceUnavailableException, MethodNotSupportedException
{
    if (listeners == null)
    {
        return;
    }

    for (int i=0; i<listeners.length; i++)
    {
        addCallListener(listeners[i]);
    }
}

/**
* Removes association between this <code>GenCall</code> and the given
* <code>GenConnection</code> object.
*
* @param connection javax.telephony.Connection
* @param cause The given cause ID.
* @param meta The given meta-code.
* @param isNewMetaEv <code>true</code> if a notification event starts of
* a new meta code group, <code>false</code> otherwise.
* @param jtapiOnly <code>true</code> if the operation is done only in the
* JTAPI layer, <code>false</code> if need update JTSPI too.
* @param opCode The given operation code.
*
* @exception javax.telephony.ResourceUnavailableException if an internal
* resource necessary for the operation is unavailable.
* @exception javax.telephony.PrivilegeViolationException if the
* application does not have the proper authority to remove the connection
* from the telephone call.
* @exception javax.telephony.InvalidStateException if some object
* required by this method is not in a valid state for this method.
* @exception javax.telephony.MethodNotSupportedException if the
* implementation does not support this method.
*/
public void removeConnection( GenConnection connection,
                             int cause, int meta, boolean isNew,
                             boolean jtapiOnly, int opCode)
    throws PrivilegeViolationException, ResourceUnavailableException,
           MethodNotSupportedException, InvalidStateException
{
    if( null == connection)
    {
        throw new IllegalArgumentException("connection is null");
    }
    synchronized(m_id)
    {
        if( m_connections.removeElement( connection ) )
        {
            int obMeta = (0 == meta ) ? Ev.META_CALL_REMOVING_PARTY : meta;
            connection.disconnect(cause, obMeta, isNew, jtapiOnly, opCode);
        }
    }
}

```

```

        if( m_connections.size() < 2 )
        {
            drop( opCode );
        }
        int otherLocals = 0;
        if( isFirstParty() )
        {
            for( int i=0; i < m_connections.size(); i++ )
            {
                if(((GenConnection)m_connections.elementAt(i)).isLocal())
                {
                    otherLocals++;
                }
            }
            if( (getState() != Call.INVALID) && ( 0 == otherLocals ) )
            {
                drop( opCode );
            }
        }
    }
}

/**
 * Disconnects the part ( leg ) of the call.
 * If the <code>Call</code> doesn't have the part, this method fails
 * silently, i.e. no part is disconnected and no exception is thrown.
 *
 * <B>Pre-conditions:</B>
 * <OL>
 * <LI>this.getProvider().getState() == Provider.IN_SERVICE
 * <LI>Let Connection c = this.getConnection( addr);
 * <LI>Let TerminalConnection tc[] = c.getTerminalConnections (see post-
 * conditions)
 * </OL>
 * <B>Post-Conditions:</B>
 * <OL>
 * <LI>this.getProvider().getState() == Provider.IN_SERVICE
 * <LI>c.getState() == Connection.DISCONNECTED
 * <LI>For all i, tc[i].getState() == TerminalConnection.DROPPED
 * <LI>c.getTerminalConnections() == null.
 * <LI>c is not an element of this.getConnections()
 * <LI><code>ConnectionEvent.ConnectionDisconnected</code> /
 * <code>ConnDisconnectedEv</code> events are delivered for the disconnected
 * Connection.
 * <LI><code>TerminalConnectionEvent.TerminalConnectionDropped</code> /
 * <code>TermConnDroppedEv</code> events are delivered for all
 * <code>TerminalConnection</code>s associated with the disconnected Connection.
 * <LI><code>ConnectionEvent.ConnectionDisconnected</code> /
 * <code>TerminalConnectionEvent.TerminalConnectionDropped</code> ;
 * <code>ConnDisconnectedEv</code> / <code>TermConnDroppedEv</code> events are
 * delivered for all other Connections and TerminalConnections dropped
 * indirectly as a result of this method.
 * <LI><code>CallEvent.CallInvalid</code> / <code>CallInvalidEv</code>, if all
 * Connections are dropped indirectly as a result of this method.
 * </OL>
 *
 * @param addr The disconnected party address name.
 * @param state The given state value.

```



```

    * @param cause The given cause ID.
    * @param meta The given meta-code.
    * @param opCode The operation code.
    */
void disconnectPart(String addr, String term, int cause, int opCode)
{
    GenConnection conn = getConnection( addr );
    if( null == conn )
    {
        return;
    }
    int meta = Ev.META_CALL_REMOVING_PARTY;
    try
    {
        if( null == term )
        {
            // Connection removing
            removeConnection( conn, cause, meta, true, true, opCode );
        }
        else
        {
            // TerminalConnection removing
            GenTerminalConnection tconn = conn.getTerminalConnection(term);
            if( null == tconn )
            {
                throw new IllegalArgumentException(
                    "the call doesn't have connection with"
                    + " specified address and terminal");
            }
            conn.removeTerminalConnection( tconn, cause, meta,
                                           true, true, opCode);
        }
    }
    catch (Exception e)
    {
        GenJtapiPeer.ms_log.warningMessage("GenCall: disconnectPart: " + e);
    }
}

/**
 * Returns <code>GenConnection</code>, associated with this
 * <code>GenCall</code> and connected to the <code>GenAddress</code> with
 * the given name. The <code>null</code> will be returned if this connection
 * cannot be found.
 *
 * @param addrName the needed connection address name.
 * @return <code>GenConnection</code> associated with this
 * <code>GenCall</code>, and with <code>GenAddress</code> which has hiven
 * name, or <code>null</code>.
 */
public GenConnection getConnection( String addrName)
{
    Address address;
    synchronized(m_connections)
    {
        int size = m_connections.size();
        for ( int i = 0; i < size; i++ )
        {

```

```

        GenConnection conn = (GenConnection)m_connections.elementAt(i);
        address = conn.getAddress();
        if( address.getName().equals( addrName ) )
        {
            return conn;
        }
    }
    return null;
}

/**
 * Returns <code>GenTerminalConnection</code>, associated with one of this
 * <code>GenCall</code> connections and connected to the given
 * <code>Terminal</code>.
 * <p>
 * The <code>null</code> will be returned if this
 * <code>GenTerminalConnection</code> cannot be found.
 *
 * @param terminal the needed TerminalConnection terminal.
 * @return <code>GenTerminalConnection</code> associated with one of this
 * <code>GenCall</code> connections, and with given <code>Terminal</code>,
 * or <code>null</code>.
 */
public GenTerminalConnection getTerminalConnection( Terminal terminal)
{
    if( null == terminal )
    {
        return null;
    }
    // TBD what about several terminalConnections to the same terminal?
    for(int i=0; i<m_connections.size(); i++)
    {
        TerminalConnection[] tc = ((Connection)m_connections.elementAt(i))
                                   .getTerminalConnections();
        for( int j=0; j<tc.length; j++)
        {
            if( terminal.equals(tc[j].getTerminal()) )
            {
                return (GenTerminalConnection)tc[j];
            }
        }
    }
    return null;
}

/**
 * Informs the call that the destination Connection was connected.
 * According to <code>Call.connect()</code> operation:
 * * "The destination <code>Connection</code> moves into the
 * <code>Connection.CONNECTED</code> state when the called party answers
 * the telephone call. If the destination Terminals are known, the answering
 * <code>TerminalConnection</code> moves into the
 * <code>TerminalConnection.ACTIVE</code> state.
 * <p>
 * <B>Events delivered to the application:</B>
 * * a <code>ConnectionEvent.CONNECTION_CONNECTED</code> /
 * <code>ConnConnectedEv</code> for the destination <code>Connection</code>
 * and a <code>TerminalConnectionEvent.TERMINAL_CONNECTION_ACTIVE</code> /

```

```

* <code>TermConnActiveEv</code> for the answering
* <code>TerminalConnection</code>, if known.
*
* @param remAddr - name of the remote address.
* @param remTerm - name of the remote terminal.
*/
protected synchronized void connected( String remAddr, String remTerm )
{
    // get destination connection [INPROGRESS | ALERTING]
    GenConnection connection = getConnection( remAddr );
    if ( null == connection )
    {
        throw new IllegalArgumentException ( "connection not found" );
    }
    if(connection.getState() != Connection.INPROGRESS &&
        connection.getState() != Connection.ALERTING)
    {
        return;
    }

    // get destination terminal connection
    GenTerminalConnection termConnection =
        connection.getTerminalConnection(remTerm);
    int observMeta = Ev.META_CALL_STARTING, observCause = Ev.CAUSE_NORMAL;
    if( connection.getState () == Connection.INPROGRESS )
    {
        // terminal connection may not exist or be IDLE
        if( null == termConnection )
        {
            // create destination terminal
            GenProvider provider = (GenProvider)getProvider();
            JtapiObjectCreator creator = provider.getObjectCreator();
            GenTerminal terminal = creator.createRemoteTerminal(
                provider, connection.getProviderPlugin(), remTerm);
            // create terminal connection
            termConnection = creator.createTerminalConnection(
                terminal, connection,
                observCause, observMeta, false );
        }
        // move new terminal connection to RINGING state
        termConnection.moveToState(TerminalConnection.RINGING, observCause,
            observMeta, false, CoreOperations.CONNECTED_CB);
        // move remote connection to ALERTING state
        connection.moveToState(Connection.ALERTING, observCause, observMeta,
            false, CoreOperations.CONNECTED_CB);
    }

    // move terminal connection to ACTIVE state
    termConnection.moveToState( TerminalConnection.ACTIVE, observCause,
        observMeta, false, CoreOperations.CONNECTED_CB);

    // move connection to CONNECTED state
    //connection.moveToState( Connection.CONNECTED, observCause, observMeta,
false,
    //
CoreOperations.CONNECTED_CB);
    // move all connections to CONNECTED state, need for extending packages.
    for(int i=0; i< m_connections.size(); i++ )
    {

```

```

        ((GenConnection)m_connections.elementAt(i)).
            moveToState( Connection.CONNECTED, observCause,
                        observMeta, false,
                        CoreOperations.CONNECTED_CB);
    }
}

/**
 * Informs the call that its local connection is offering.
 *
 * @param localAddress local address
 * @param remAddrName calling address
 * @param remTermName calling terminal
 * @param loopback <code>true</code> if the local and remote addresses belong to
 * the same provider, <code>false</code> otherwise.
 * @param plug the <code>ProviderPlugin</code> which get the incoming call.
 */
protected void offering( GenAddress localAddress, String remAddrName,
                        String remTermName, boolean loopback,
                        ProviderPlugin plug)
{
    try
    {
        addObservers(localAddress.getCallObservers());
        addCallListeners(localAddress.getCallListeners());
        GenTerminal[] terminals = (GenTerminal[])localAddress.getTerminals();
        if( null != terminals )
        {
            for(int i=0; i< terminals.length; i++)
            {
                addObservers(terminals[i].getCallObservers());
                addCallListeners(terminals[i].getCallListeners());
            }
        }
        int observCause = Ev.CAUSE_NEW_CALL, observMeta = Ev.META_CALL_STARTING;
        moveToState(Call.ACTIVE, observCause, observMeta, true,
                    CoreOperations.OFFERING_CB );
        JtapiObjectCreator creator = m_provider.getObjectCreator();
        GenConnection localConnection, remoteConnection;
        if( loopback )
        {
            localConnection = getConnection( localAddress.getName() );
            if( null == localConnection )
            {
                localConnection = creator.createConnection( this,
                                                            localAddress, observCause, observMeta, false);
            }
            remoteConnection = getConnection( remAddrName );
            remoteConnection.moveToState( Connection.INPROGRESS, observCause,
                                         observMeta, false, CoreOperations.OFFERING_CB);
        }
        else
        {
            GenAddress remoteAddress =
                creator.createRemoteAddress(m_provider, plug, remAddrName);
            GenTerminal remoteTerminal =
                creator.createRemoteTerminal(m_provider, plug, remTermName);
            remoteAddress.addTerminal(remoteTerminal);
            setOriginatingEndpoint(remoteAddress,remoteTerminal);
        }
    }
}

```

```

        localConnection = creator.createConnection( this,
            localAddress, observCause, observMeta, false);
        remoteConnection = creator.createConnection( this,
            remoteAddress, observCause, observMeta, false);

        remoteConnection.moveToState( Connection.INPROGRESS,
            observCause, observMeta, false,
            CoreOperations.OFFERING_CB);

        GenTerminalConnection remTermConnection =
        creator.createTerminalConnection( remoteTerminal, remoteConnection,
            observCause, observMeta, false);
        remTermConnection.moveToState( TerminalConnection.ACTIVE,
            observCause, observMeta, false,
            CoreOperations.OFFERING_CB);
    }
    remoteConnection.moveToState( Connection.CONNECTED, observCause,
        observMeta, false,
        CoreOperations.OFFERING_CB);
    localConnectionOffering( localConnection);
} catch (Exception e)
{
    // TBD need roolback
    GenJtapiPeer.ms_log.errorMessage( "GenCall: offering: " + e );
}
}

/**
 * Informs the <code>GenCall</code> that distination part is ringing.
 * The destination <code>Connection</code> moves into the
 * <code>Connection.ALERTING</code> state.
 * <code>TerminalConnection</code> object may be created to model
 * the relationship between any known destination Terminals associated with
 * the Call, each in the <code>TerminalConnection.RINGING</code> state.
 * If the destination Terminals are unknown , then no TerminalConnections are
 * created.
 * <p>
 * <B>Events delivered to the application:</B>
 * a <code>ConnectionEvent.CONNECTION_ALERTING</code> /
 * <code>ConnAlertingEv</code> for the destination <code>Connection</code>,
 * a <code>TerminalConnectionEvent.TERMINAL_CONNECTION_CREATED</code> /
 * <code>TermConnCreatedEv</code> and
 * <code>TerminalConnectionEvent.TERMINAL_CONNECTION_RINGING</code> /
 * <code>TermConnRingingEv</code> for
 * any destination TerminalConnections created.
 *
 * @param addrName name of distination address
 * @param termName name of distination terminal, may be <code>null</code>
 */
protected void ringing( String addrName, String termName)
{
    // synchronize the operation
    int meta = Ev.META_CALL_STARTING, cause = Ev.CAUSE_NORMAL;
    synchronized ( m_id )
    {
        // get remote connection [IDLE | INPROGRESS]
        GenConnection connection = getConnection( addrName );
        if ( null == connection )
        {

```

```

        throw new IllegalArgumentException( "connection not found" );
    }
    // get remote terminal connection
    if( null != termName)
    {
        GenTerminalConnection termConnection =
            connection.getTerminalConnection( termName);
        if( null == termConnection )
        {
            JtapiObjectCreator creator = m_provider.getObjectCreator();
            // create destination terminal
            GenTerminal term = null;
            if( m_provider.isLocalTerminal( termName ) )
            {
                try
                {
                    term =
(GenTerminal)m_provider.getTerminal(termName);
                }
                catch(InvalidArgumentException e)
                {
                    // we cannot be here
                }
            }
            else
            {
                term = creator.createRemoteTerminal(m_provider,
                    connection.getProviderPlugin(), termName
                );
            }
            // create terminal connection
            termConnection = creator.createTerminalConnection(term,
                connection, cause, meta, false );
        }
        // move terminal connection to RINGING state
        termConnection.moveToState( TerminalConnection.RINGING,
            cause, meta, false,
            CoreOperations.RINGING_CB);
    }
    // move objects to correct states
    // move connection to ALERTING state
    connection.moveToState( Connection.ALERTING, cause,
        meta, false, CoreOperations.RINGING_CB);
    synchronized( m_connections )
    {
        int size = m_connections.size();
        GenConnection conn;
        for(int i=0; i<size; i++)
        {
            conn = (GenConnection)m_connections.elementAt(i);
            if( ! conn.equals(connection) )
            {
                conn.moveToState(Connection.CONNECTED,
                    cause, meta, false, CoreOperations.RINGING_CB);
            }
        }
    }
}

```

```

    }
}

/**
 * Part of offering operation. The method is overwritten by the extend package.
 */
protected void localConnectionOffering( GenConnection connection)
{
    Address localAddress = connection.getAddress();
    GenTerminal[] terminals = (GenTerminal[])localAddress.getTerminals();
    int observCause = Ev.CAUSE_NEW_CALL, observMeta = Ev.META_CALL_STARTING;
    GenTerminalConnection[] localTermConnections =
        new GenTerminalConnection[terminals.length];
    JtapiObjectCreator creator = m_provider.getObjectCreator();
    for(int i=0; i<terminals.length; i++)
    {
        localTermConnections[i] =
            creator.createTerminalConnection( terminals[i],
                connection, observCause, observMeta, false );
        localTermConnections[i].moveToState(TerminalConnection.RINGING,
            observCause, observMeta, false,
            CoreOperations.OFFERING_CB);
    }
    connection.moveToState( Connection.ALERTING, observCause,
        observMeta, false,
        CoreOperations.OFFERING_CB);
}

/**
 * Sets parameters of the originating endpoint for this <code>Call</code>.
 *
 * @param origAddress The given originating <code>Address</code>.
 * @param origTerminal The given originating <code>Terminal</code>.
 */
protected void setOriginatingEndpoint( GenAddress address,
    GenTerminal terminal)
{
    m_origAddress = address;
    m_origTerminal = terminal;
}

/**
 * Delivers the <code>CallObservationEndedEv</code> event to a given
 * <code>CallObserver</code> object, which has been removed from this
 * <code>GenCall</code>.
 *
 * @param observer The given <code>CallObserver</code> which has been
 * removed.
 */
protected void notifyObserverRemoved( CallObserver observer )
{
    CallObserver[] obList = { observer };
    CallEv[] evList = { new GenCallObservationEndedEv(this) };
    JtapiObjectNotifier.getInstance().enqueue( obList, evList );
}

```

```

}

/**
 * Delivers the <code>CallEvent.CALL_EVENT_TRANSMISSION_ENDED</code>
 * event to a given <code>CallListener</code> object, which has been
 * removed from this <code>GenCall</code>.
 *
 * @param listener The given <code>CallListener</code> which has been
 * removed.
 */
protected void notifyListenerRemoved(CallListener listener)
{
    CallListener[] lsList = { listener };
    CallEvent[] evList = { new GenCallEvent(this,
        CallEvent.CALL_EVENT_TRANSMISSION_ENDED, CallEvent.CAUSE_UNKNOWN) };
    JtapiObjectNotifier.getInstance().enqueue( lsList, evList );
}

/**
 * Adds the given <code>Connection</code> to this <code>GenCall</code>.
 *
 * @param connection The added Connection.
 */
protected void addConnection(Connection conn)
{
    synchronized(m_connections) {
        if(null == conn || m_connections.contains(conn))
        {
            return;
        }
        m_connections.addElement(conn);
    }
}

protected void finalize() throws Throwable
{
    //notify all observers
    CallObserver[] obList = getObservers();
    CallEv[] evList = { new GenCallObservationEndedEv( this ) };
    JtapiObjectNotifier.getInstance().enqueue( obList, evList );
    m_observers.removeAllElements();
    //notify all listeners
    CallListener[] lsList = getCallListeners();
    CallEvent[] eventsList = {
        new GenCallEvent(this,
            CallEvent.CALL_EVENT_TRANSMISSION_ENDED,
            CallEvent.CAUSE_UNKNOWN) };
    JtapiObjectNotifier.getInstance().enqueue( lsList, eventsList );
    m_listeners.removeAllElements();
    super.finalize();
}
}

```



```

package com.ibm.hrl.jtapi;

/*
 * (c) Copyright IBM Corporation 1998,1999
 * IBM Research Laboratory in Haifa
 * Generic JTAPI Implementation (JTAPI 1.2 / JTAPI 1.3)
 * -----
 * Package      : com.ibm.hrl.jtapi
 * Class        : GenJtapiPeer
 * Created      : ██████████
 */

import java.util.Properties;
import java.util.Enumeration;
import java.util.StringTokenizer;
import java.util.Hashtable;
import java.util.Vector;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import javax.telephony.*;

import com.ibm.hrl.jtapi.jtspi.ProviderPlugin;
import com.ibm.hrl.jtapi.jtspi.CallControlProviderPlugin;
import com.ibm.hrl.jtapi.jtspi.CallCenterProviderPlugin;
import com.ibm.hrl.jtapi.jtspi.MediaProviderPlugin;

import com.ibm.hrl.jtapi.util.JtapiObjectCreator;
import com.ibm.hrl.util.SystemLog;
import com.ibm.hrl.util.DebugLog;

/**
 * Generic implementation of the
 * JtapiPeer interface.
 *
 * @see javax.telephony.JtapiPeer
 * @see com.ibm.hrl.jtapi.GenProvider
 *
 * @author Alexey Roytman
 */

public class GenJtapiPeer implements JtapiPeer
{
    // IBM Copyright
    public static final String IBM_Copyright = Copyright.SHORT_STRING;

    /*
     * Configuration file, with following (properties) format:
     * service = fileName.
     * Where service is name one of the peer services and
     * fileName is name of the service configuration file.
     */
    private static String ms_configFileName = "genjtapi.cfg";

    private static String ms_addressString      = "address.";
    private static char ms_delimiter            = ';';

```

```

private static String ms_eventsViewable      = "GenJtapiEventsViewable";

/**
 * Hashtable of created providers.
 * keys: strings from getProviderMethod
 * values: created providers
 */
private static Hashtable providers;

/*
 * Possible services and corresponding configuration file names
 *
 * key - service name
 * data - configuration file name
 */
private Properties m_services;

// We can use it from all parts of the program
// The PROGRESS_3 logger reserved for events
static public DebugLog ms_log;

// ctors .....

static {
    // should be changed for MOBILE
    ms_log = new SystemLog();
    ms_log.setLogEnabled( DebugLog.ERROR | DebugLog.WARNING |
                          DebugLog.PROGRESS, true);
    providers = new Hashtable();
}

/**
 * The default <code>GenJtapiPeer</code> constructor.
 * Initializes the Services properties.
 *
 * @exception javax.telephony.JtapiPeerUnavailableException if the
 * GenJTAPI Peer Toolkit cannot be initialized.
 */
public GenJtapiPeer() throws JtapiPeerUnavailableException {

    try {

        // read from configuration file into Properties object
        m_services = new Properties();
        FileInputStream fs = new FileInputStream(ms_configFileName);
        m_services.load(fs);
        fs.close();
    } catch (FileNotFoundException fe) {
        ms_log.errorMessage("Cannot found the " + ms_configFileName +
                           " file");
        throw new JtapiPeerUnavailableException(fe.getMessage());
    } catch (IOException ioe) {
        ms_log.errorMessage("Cannot load configuration properties", ioe);
        throw new JtapiPeerUnavailableException(ioe.getMessage());
    }
}

```

```

// JTAPI methods .....

/**
 * Returns the full class name of this <code>JtapiPeer</code> instance.
 * <p>
 * This method implements the
 * <code>javax.telephony.JtapiPeer.getName()</code> method.
 *
 * @return The full class name of this <code>JtapiPeer</code> instance.
 *
 * @see javax.telephony.JtapiPeer.getName()
 */
public String getName () {
    return getClass().getName();
}

/**
 * Returns <code>Provider</code> object given a string argument which contains
 * the desired service specification. If <code>Provider</code> with given
 * the string argument was created, next calls the method with the <b>same</b>
 * argument return the references to the same object.<br>
 * Note: may be we need compare formatted string argument.
 * <p>
 * This method implements the
 * <code>javax.telephony.JtapiPeer.getProvider()</code> method.
 * <p>
 * The <code>providerString</code> argument has the following format:
 * <br><code>service_name; arg1=val1; ...</code>
 * <br> where <code>service_name</code> is mandatory and each optional
 * argument pair which follows is separated by a semi-colon.
 * <p>
 * Possible argument names are as follows:
 * <ul>
 * <li><code>login</code> - provides the login user name to the
 * <code>Provider</code>
 * <li><code>passwd</code> - provides a password to the <b>Provider</b>
 * <li>other optional arguments.
 * </ul>
 * All arguments are passed to the specific implementation of the
 * <code>ProviderService</code> object.
 * <p>
 * If the <code>providerString</code> argument is <code>null</code>, this method
 * returns the default <code>Provider</code> as determined by the
 * <code>GenJtapiPeer</code> implementation.
 * <p>
 * <b>Post-conditions:</b>
 * <ol>
 * <li>this.getProvider().getState() == Provider.OUT_OF_SERVICE
 * </ol>
 *
 * @return The desired instance of the <code>Provider</code> object.
 *
 * @param providerString Specification of the desired service.
 *
 * @exception javax.telephony.ProviderUnavailableException indicates a
 * <code>Provider</code> corresponding to the given string is unavailable.
 *
 * @see javax.telephony.JtapiPeer.getProvider()
 */
public Provider getProvider( String providerString )

```

```

        throws ProviderUnavailableException {

ms_log.progressMessage( "JtapiPeer.getProvider(" +
                        providerString + ")");
if( null == providerString ) {
    throw new ProviderUnavailableException(
        "Meantime we don't support default provider");
}
ms_log.progressMessage( "\nJtapiPeer.getProvider( before synchronized in
");
synchronized ( providers ) {
    ms_log.progressMessage( "\nJtapiPeer.getProvider( synchronized in ");
    // TBD if this is right??
    if(providers.containsKey(providerString)) {
        return (Provider)providers.get(providerString);
    }
    ProviderSpec spec = parseProviderString( providerString );
    Hashtable providerTable = new Hashtable();
    int level = initPlugins( (String)m_services.get(spec.name),
                            spec.initData, providerTable);
    if( providerTable.size() == 0) {
        ms_log.errorMessage("JtapiPeer: cannot initialize any plugin,
bye!");
        throw new ProviderUnavailableException( "cannot initialize any
plugin");
    }
    ms_log.progressMessage( "JtapiPeer: we work with level " + level);
    JtapiObjectCreator creator = new JtapiObjectCreator(level);
    Provider provider = creator.createProvider( spec.name, providerTable,
this);
    providers.put(providerString, provider);
    ms_log.progressMessage( "\nJtapiPeer.getProvider( synchronized out ");
    return provider;
}

}

/**
 * Returns the services that this implementation supports.
 * <p>
 * This method implements the
 * <code>javax.telephony.JtapiPeer.getServices()</code> method.
 *
 * @return The services that this implementation supports.
 *
 * @see javax.telephony.JtapiPeer.
 */
public String[] getServices () {
    String[] services = new String[m_services.size()];
    Enumeration e = m_services.keys();
    for( int i=0; e.hasMoreElements(); i++) {
        services[i] = (String)e.nextElement();
    }
    return services;
}

// Implementation methods .....

/**

```

```

* Parses a given provider string and puts specifications of the desired
* service provider to the <b>ProviderSpec</b> structure.
*
* @return The resulting service provider specifications.
*
* @param providerString The given provider string.
*
* @exception javax.telephony.ProviderUnavailableException indicates the
* service name is not specified in the provider string.
*/
private ProviderSpec parseProviderString( String providerString )
    throws ProviderUnavailableException {

    ProviderSpec provSpec = new ProviderSpec();
    StringTokenizer list = new StringTokenizer( providerString, ";" );
    while( list.hasMoreTokens() ) {
        String token = list.nextToken();
        if( null == provSpec.name ) {
            // the first token should be 'service_name'
            provSpec.name = token.trim();
        } else {
            // put the "tag = value" pair
            if( null == provSpec.initData ) {
                provSpec.initData = new Properties();
            }
            int index = token.indexOf( '=' );
            if( index > 0 ) {
                String key = token.substring( 0, index ).trim();
                if( key.length() > 0 ) {
                    String value = token.substring( index + 1 ).trim();
                    if( value.length() > 0 )
                        provSpec.initData.put( key, value );
                }
            }
        }
    }

    if ( null == provSpec.name || !m_services.containsKey(provSpec.name) ) {
        throw new ProviderUnavailableException (
            ProviderUnavailableException.CAUSE_INVALID_SERVICE,
            "invalid service specification" );
    }

    return provSpec;
}

// private container class
private class ProviderSpec {
    String name;
    Properties initData;
}

protected int initPlugins(final String cnfFileName,
                           final Properties initData,
                           Hashtable pluginsTable ) {

    Properties pluginsProp = new Properties();
    try {
        FileInputStream fs = new FileInputStream(cnfFileName);
        pluginsProp.load(fs);
    }
}

```

```

        fs.close();
    } catch ( Exception e) {
        ms_log.errorMessage("GenJtapiPeer: initPlugins : ", e);
        System.exit(1);
    }
    Enumeration pluginsEnum = pluginsProp.keys();
    // Hashtable with keys: - plugins objects, data: - hashtable2
    // Hashtable2 keys: addresses' names, data: - array of associated terminals'
names
    int level = 0;
    while(pluginsEnum.hasMoreElements()) {
        String plugName = (String)pluginsEnum.nextElement();
        String plugData = pluginsProp.getProperty(plugName);
        int delimIndex = plugData.indexOf(ms_delimiter);
        if( 1 > delimIndex || delimIndex == (plugData.length() - 1)) {
            continue;
        }
        try {
            Class pluginClass = Class.forName(plugData.substring(0, delimIndex));
            ProviderPlugin plugin = (ProviderPlugin)pluginClass.newInstance();
            Properties plugResources = new Properties();
            FileInputStream fs = new FileInputStream( plugData.substring(delimIndex
+1));

            plugResources.load(fs);
            fs.close();
            Enumeration resourcesEnum = plugResources.keys();
            Hashtable addressTerm = new Hashtable();
            while(resourcesEnum.hasMoreElements()) {
                String resourceName = (String)resourcesEnum.nextElement();
                if(resourceName.startsWith(ms_addressString)) {
                    Vector terminalsVector = new Vector();
                    String termList = plugResources.getProperty(resourceName);
                    plugResources.remove(resourceName);
                    int delim, lastDelim = 0;
                    while( lastDelim < termList.length() ) {
                        delim = termList.indexOf(ms_delimiter, lastDelim);
                        if( -1 != delim) {
                            terminalsVector.addElement(termList.substring(lastDelim, delim));
                            lastDelim = delim + 1;
                        } else {
                            terminalsVector.addElement(termList.substring(lastDelim));
                            lastDelim = termList.length();
                        }
                    }
                    if(0 == terminalsVector.size()) {
                        continue;
                    }
                    String[] terminalsArray = new String[terminalsVector.size()];
                    terminalsVector.copyInto(terminalsArray);
                    addressTerm.put(resourceName.substring(ms_addressString.length()), terminalsArray);
                }
            }
            if(0 == addressTerm.size()) {
                continue;
            }
            if( null != initData ) {

```

```

        resourcesEnum = initData.keys();
        while(resourcesEnum.hasMoreElements()) {
            String propert = (String)resourcesEnum.nextElement();
            plugResources.put(propert, initData.get(propert));
        }
    }
    if( plugResources.containsKey(ms_eventsViewable) ){
        Boolean b = new
Boolean((String)plugResources.get(ms_eventsViewable));
        boolean viewEvents = b.booleanValue();
        ms_log.setLogEnabled( DebugLog.PROGRESS_3, viewEvents);
        plugResources.remove(ms_eventsViewable);
    }
    plugin.prvInitialize(addressTerm, plugResources);
    level |= checkSupportedLevel(plugin);
    // updateCapabilities(plugin);
    pluginsTable.put(plugin, addressTerm);
} catch (Exception e) {
    ms_log.warningMessage("GenJtapiPeer: initPlugins : ", e);
    // We don't use the plugin
}
}
return level;
}

protected int checkSupportedLevel(ProviderPlugin plugin) {
    int level = plugin.prvSupportedLevel();
    if((0 != (level & ProviderPlugin.CALLCONTROL_LEVEL)) &&
        ! (plugin instanceof CallControlProviderPlugin)) {
        level &= ~ProviderPlugin.CALLCONTROL_LEVEL;
    } else if (( 0 != (level & ProviderPlugin.CALLCENTER_LEVEL)) &&
        ! (plugin instanceof CallCenterProviderPlugin)) {
        level &= ~ProviderPlugin.CALLCENTER_LEVEL;
    } else if (( 0 != (level & ProviderPlugin.MEDIA_12_LEVEL)) &&
        ! (plugin instanceof MediaProviderPlugin)) {
        level &= ~ProviderPlugin.MEDIA_12_LEVEL;
    }
    return level;
}
}
}

```

```
package com.ibm.hrl.jtapi.jtsmi;
```

```
/*
```

```
 * (c) Copyright IBM Corporation 1998,1999,2000
```

```
 * IBM Research Laboratory in Haifa
```

```
 * Generic JTAPI Implementation (JTAPI 1.3)
```

```
 * -----
```

```
 * Package   : com.ibm.hrl.jtapi.jtsmi
```

```
 * Class     : JtsmiPlugin
```

```
 * Created   : [REDACTED]
```

```
 */
```

```
import java.util.Properties;
```

```
import java.util.Hashtable;
```

```
public interface JtsmiPlugin
```

```
{
```

```
    public void init( String ruleFile, Properties initData );
```

```
    public String addressConvert( String address );
```

```
    public String pluginForAddress( String address );
```

```
    public boolean isInService( Hashtable providerPluginStates );
```

```
}
```



```

package com.ibm.hrl.jtapi.jtsmi;

/*
 * (c) Copyright IBM Corporation 1998,1999,2000
 * IBM Research Laboratory in Haifa
 * Generic JTAPI Implementation (JTAPI 1.3)
 * -----
 * Package      : com.ibm.hrl.jtapi.jtsmi
 * Class        : MultyPluginJTSMI
 * Created      : ██████████
 */

import java.util.Hashtable;
import java.util.Enumeration;
import java.util.Properties;
import java.util.Vector;

import java.io.InputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import javax.telephony.Address;
import javax.telephony.Provider;
import javax.telephony.Terminal;
import javax.telephony.Call;
import javax.telephony.ResourceUnavailableException;
import javax.telephony.InvalidArgumentException;
import javax.telephony.PlatformException;

import com.ibm.hrl.jtapi.GenAddress;
import com.ibm.hrl.jtapi.GenTerminal;
import com.ibm.hrl.jtapi.GenProvider;
import com.ibm.hrl.jtapi.GenJtapiPeer;
import com.ibm.hrl.jtapi.GenConnection;

import com.ibm.hrl.jtapi.capabilities.GenAddressCapabilities;
import com.ibm.hrl.jtapi.capabilities.GenCallCapabilities;
import com.ibm.hrl.jtapi.capabilities.GenConnectionCapabilities;
import com.ibm.hrl.jtapi.capabilities.GenProviderCapabilities;
import com.ibm.hrl.jtapi.capabilities.GenTerminalCapabilities;
import com.ibm.hrl.jtapi.capabilities.GenTerminalConnectionCapabilities;

import com.ibm.hrl.jtapi.util.JtapiObjectCreator;

import com.ibm.hrl.jtapi.jtspi.JtapiCallbacks;
import com.ibm.hrl.jtapi.jtspi.ProviderPlugin;

public class MultyPluginJTSMI extends Jtsmi
{
    // IBM Copyright
    public static final String IBM_Copyright    = Copyright.SHORT_STRING;

    public static String ms_jtsmiName = "JtsmiPlugin";

    // providers' plugins objects
    // key - plugin name
    // value - plugin object
    private Hashtable m_plugins;

```

```

// plugin's states
// key plugin object
// value Boolean state
private Hashtable m_pluginStates;

// key - addressName
// value terminal name array.
private Hashtable m_resources;

private JtsmiPlugin m_jtsmiPlugin;

private static Boolean ms_inService = new Boolean( true );
private static Boolean ms_outOfService = new Boolean( false );

public MultyPluginJTSMI( Properties pluginsProp, Properties initData )
{
    m_resources = new Hashtable();
    m_plugins = new Hashtable(3);
    m_pluginStates = new Hashtable(3);
    doIt( pluginsProp, initData );
}

private synchronized void doIt( Properties pluginsProp, final Properties initData
)
{
    Enumeration pluginsEnum = pluginsProp.keys();
    int runningThreads = 0;
    final Vector plugins = new Vector();
    while(pluginsEnum.hasMoreElements())
    {
        final String plugName = (String)pluginsEnum.nextElement();
        String plugData = pluginsProp.getProperty(plugName);
        int delimIndex = plugData.indexOf(ms_delimiter);
        if( 1 > delimIndex || delimIndex == (plugData.length() - 1))
        {
            continue;
        }
        final ProviderPlugin plugin;
        final String resourceFile;
        try
        {
            Class pluginClass = Class.forName(plugData.substring(0, delimIndex));
            if( plugName.equals(ms_jtsmiName))
            {
                m_jtsmiPlugin = (JtsmiPlugin)pluginClass.newInstance();
                m_jtsmiPlugin.init( plugData.substring(delimIndex + 1), initData );
                continue;
            }
            plugin = (ProviderPlugin)pluginClass.newInstance();
            resourceFile = plugData.substring(delimIndex + 1);
        }
        catch(Exception e)
        {
            GenJtapiPeer.ms_log.warningMessage("GenJtapiPeer: initPlugin - " +
plugName, e);
            // We don't use the plugin
            continue;
        }
    }
}

```

```

    }
    runningThreads ++;
    Runnable r = new Runnable()
    {
        public void run()
        {
            Hashtable table;
            try
            {
                table = initPlugin( plugin, resourceFile, initData );
                m_resources.put( plugin, table );
                m_plugins.put( plugName, plugin );
                m_pluginStates.put( plugin, ms_inService );
            }
            catch ( Exception e )
            {
                // We don't use the plugin
                GenJtapiPeer.ms_log.warningMessage("GenJtapiPeer: initPlugin -
" + plugName, e);

                m_resources.put( plugin, new Hashtable());
            }
            synchronized( MultyPluginJTSMI.this )
            {
                MultyPluginJTSMI.this.notify();
            }
        }
    };
    GenJtapiPeer.runIt(r);
}
while( runningThreads != m_resources.size() )
{
    try {
        wait();
    }
    catch (InterruptedException e )
    {}
}
int level = -1; // 111111111111....
pluginsEnum = m_plugins.elements();
while( pluginsEnum.hasMoreElements() )
{
    level &= ((ProviderPlugin)pluginsEnum.nextElement()).prvSupportedLevel();
}
m_creator = new JtapiObjectCreator( level );
}

public void init( GenProvider provider) throws InvalidArgumentException
{
    m_provider = provider;
    // Need update callback creator.
    JtapiCallbacks callbacks =
        m_creator.createJtapiCallbacks(provider, this);
    Enumeration plugins = m_plugins.elements();
    ProviderPlugin plugin;
    while( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.prvSetCallback(callbacks);
    }
}

```

```

        createEndpoints( plugin, provider, (Hashtable)m_resources.get( plugin ));
    }
    m_resources = null;
    changeProviderState();
}

public void updateAddressCapabilities(
    GenAddressCapabilities capabilities)
{
    GenAddressCapabilities tmpCapabilities = new GenAddressCapabilities();
    Enumeration plugins = m_plugins.elements();
    ProviderPlugin plugin;
    if( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateAddressCapabilities( capabilities );
    }
    while( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateAddressCapabilities( tmpCapabilities );
        capabilities.and( tmpCapabilities );
        tmpCapabilities.reset();
    }
}

/**
 * Updates <code>static</code> <code>GenTerminalCapabilities</code> for this
 * provider plugin. The <code>GenTerminalCapabilities</code> implements the
 * capability interfaces for the <code>Terminal</code> object from the JTAPI
 * core and all extension packages. By default all methods of instance of the
 * class, except <code>isObservable()</code>, return <code>false</code>.
 * By this method the plugin can change the default behaved of the instance.
 */
public void updateTerminalCapabilities(
    GenTerminalCapabilities capabilities)
{
    GenTerminalCapabilities tmpCapabilities = new GenTerminalCapabilities();
    Enumeration plugins = m_plugins.elements();
    ProviderPlugin plugin;
    if( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateTerminalCapabilities( capabilities );
    }
    while( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateTerminalCapabilities( tmpCapabilities );
        capabilities.and( tmpCapabilities );
        tmpCapabilities.reset();
    }
}

/**
 * Updates <code>static</code> <code>GenTerminalConnectionCapabilities</code>

```

```

* for this provider plugin.
* The <code>GenTerminalConnectionCapabilities</code> implements the
* capability interfaces for the <code>TerminalConnection</code> object from
* the JTAPI core and all extension packages. By default all methods of
* instance of the class, except <code>canAnswer()</code>, return
* <code>>false</code>. By this method the plugin can change the default
* behaved of the instance.
*
*/
public void updateTerminalConnectionCapabilities(
    GenTerminalConnectionCapabilities capabilities)
{
    GenTerminalConnectionCapabilities tmpCapabilities =
        new GenTerminalConnectionCapabilities();
    Enumeration plugins = m_plugins.elements();
    ProviderPlugin plugin;
    if( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateTerminalConnectionCapabilities( capabilities );
    }
    while( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateTerminalConnectionCapabilities( tmpCapabilities );
        capabilities.and( tmpCapabilities );
        tmpCapabilities.reset();
    }
}

/**
* Updates <code>static</code> <code>GenConnectionCapabilities</code> for
* this provider plugin. The <code>GenConnectionCapabilities</code>
* implements the capability interfaces for the <code>Connection</code>
* object from the JTAPI core and all extension packages. By default all
* methods of instance of the class, except <code>canDisconnect()</code>,
* return <code>>false</code>. By this method the plugin can change the
* default behaved of the instance.
*
*/
public void updateConnectionCapabilities(
    GenConnectionCapabilities capabilities)
{
    GenConnectionCapabilities tmpCapabilities =
        new GenConnectionCapabilities();
    Enumeration plugins = m_plugins.elements();
    ProviderPlugin plugin;
    if( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateConnectionCapabilities( capabilities );
    }
    while( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateConnectionCapabilities( tmpCapabilities );
        capabilities.and( tmpCapabilities );
        tmpCapabilities.reset();
    }
}

```

```

}

/**
 * Updates <code>static</code> <code>GenACallCapabilities</code> for this
 * provider plugin. The <code>GenCallCapabilities</code> implements the
 * capability interfaces for the <code>Call</code> object from the JTAPI
 * core and all extension packages. By default all methods of instance of the
 * class, except <code>isObservable()</code> and <code>canConnect()</code>,
 * return <code>false</code>. By this method the plugin can change the
 * default behaved of the instance.
 *
 */
public void updateCallCapabilities( GenCallCapabilities capabilities)
{
    GenCallCapabilities tmpCapabilities =
        new GenCallCapabilities();
    Enumeration plugins = m_plugins.elements();
    ProviderPlugin plugin;
    if( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateCallCapabilities( capabilities );
    }
    while( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateCallCapabilities( tmpCapabilities );
        capabilities.and( tmpCapabilities );
        tmpCapabilities.reset();
    }
}

/**
 * Updates <code>static</code> <code>GenProviderCapabilities</code> for this
 * provider plugin. The <code>GenProviderCapabilities</code> implements the
 * capability interfaces for the <code>Providers</code> object from the JTAPI
 * core and all extension packages. By default all methods of instance of the
 * class, except <code>isObservable()</code>, return <code>false</code>.
 * By this method the plugin can change the default behaved of the instance.
 *
 */
public void updateProviderCapabilities( GenProviderCapabilities capabilities)
{
    GenProviderCapabilities tmpCapabilities =
        new GenProviderCapabilities();
    Enumeration plugins = m_plugins.elements();
    ProviderPlugin plugin;
    if( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateProviderCapabilities( capabilities );
    }
    while( plugins.hasMoreElements() )
    {
        plugin = (ProviderPlugin)plugins.nextElement();
        plugin.updateProviderCapabilities( tmpCapabilities );
        capabilities.and( tmpCapabilities );
        tmpCapabilities.reset();
    }
}

```

```

    }
}

public GenAddress getGenAddress( String addressName )
                                throws InvalidArgumentException
{
    addressName = m_jtspiPlugin.addressConvert( addressName );
    GenAddress address = getAddress( addressName );
    ProviderPlugin plugin;
    Boolean state;
    if( null == address )
    {
        String providerPluginName = m_jtspiPlugin.pluginForAddress( addressName
);
        plugin = (ProviderPlugin)m_plugins.get( providerPluginName );
        if( plugin == null )
        {
            throw new PlatformException(
                "MultyPluginJTSMI.getGenAddress: invalid plugin name ");
        }
        state = (Boolean)m_pluginStates.get( plugin);
        if( !state.booleanValue() )
        {
            throw new PlatformException(" The provider plugin is out of
service");
        }
        address = m_creator.createRemoteAddress(m_provider, plugin, addressName);
    }
    else
    {
        plugin = address.getProviderPlugin();
        state = (Boolean)m_pluginStates.get( plugin);
        if( !state.booleanValue() )
        {
            throw new PlatformException(" The provider plugin is out of
service");
        }
    }
    return address;
}

public GenAddress getAddress( String addressName )
                                throws InvalidArgumentException
{
    addressName = m_jtspiPlugin.addressConvert( addressName );
    if( addressName == null )
    {
        throw new InvalidArgumentException( "null address number" );
    }
    return (GenAddress)m_addresses.get( addressName );
}

public GenTerminal getGenTerminal( String terminalName )
                                throws InvalidArgumentException
{
    GenTerminal terminal = getTerminal( terminalName );
    /* if( null == terminal )
    {

```

```

        terminal = m_creator.createRemoteTerminal(m_provider, m_plugin,
terminalName);
    } */
    return terminal;
}

public synchronized void inService(boolean state, ProviderPlugin plugin)
    throws IllegalStateException, IllegalArgumentException
{
    if( null == plugin )
    {
        throw new IllegalArgumentException( " null plugin object " );
    }
    Boolean newState = ( state ) ? ms_inService : ms_outOfService;
    Boolean oldState = (Boolean)m_pluginStates.get(plugin);
    if( !newState.equals(oldState) )
    {
        m_pluginStates.put( plugin, newState );
        changeProviderState();
    }
    if( ! state )
    {
        try
        {
            Call[] calls = m_provider.getCalls();
            if( calls != null)
            {
                for( int i=0; i<calls.length; i++)
                {
                    GenConnection[] conn =
                        (GenConnection[])calls[i].getConnections();
                    for( int j=0; j< conn.length; j++)
                    {
                        if( conn[j].getProviderPlugin().equals(plugin))
                        {
                            conn[j].disconnect();
                        }
                    }
                }
            }
        }
        catch ( Exception e)
        {
            GenJtapiPeer.ms_log.errorMessage(
                "MultyPluginJTSMI.inService " , e);
        }
    }
}

private void changeProviderState()
{
    Hashtable table = new Hashtable( m_plugins.size() );
    Enumeration plugNames = m_plugins.keys();
    String pluginName;
    while( plugNames.hasMoreElements() )
    {
        pluginName = (String)plugNames.nextElement();
        table.put( pluginName,
            m_pluginStates.get( m_plugins.get(pluginName)));
    }
}

```



```

    }
    int state = m_provider.getState();
    // we cannot change the SHUTDOWN state of provider.
    if( m_jtsmiPlugin.isInService( table ))
    {
        if( state == Provider.OUT_OF_SERVICE )
        {
            m_provider.moveToState( Provider.IN_SERVICE );
        }
    }
    else
    {
        if( state == Provider.IN_SERVICE )
        {
            m_provider.moveToState( Provider.OUT_OF_SERVICE );
        }
    }
}

```

```

public boolean isPluginInService( ProviderPlugin plugin)
{
    Boolean state = (Boolean)m_pluginStates.get(plugin);
    if( state != null )
    {
        return state.booleanValue();
    }
    return false;
}
}

```

```

package com.ibm.hrl.jtapi.jtspi;

/*
 * (c) Copyright IBM Corporation 1998,1999
 * IBM Research Laboratory in Haifa
 * Generic JTAPI Implementation (JTAPI 1.2)
 * -----
 * Package          : com.ibm.hrl.jtapi.jtspi
 * Abstract class   : ProviderPlugin
 * Created          : ██████████
 */

import java.util.Hashtable;
import java.util.Properties;

import com.ibm.hrl.jtapi.GenProvider;

/**
 * Defines methods of provider-specific services for the core package.
 * <p>
 * The provider-specific implementation of the interface should be registered
 * in the service configuration file.
 *
 * The JTSPI implementor is not required to know the GenJTAPI implementation.
 *
 * @see com.ibm.hrl.jtapi.GenJtapiPeer#getProvider
 * @see com.ibm.hrl.jtapi.GenProvider
 * @see com.ibm.hrl.jtapi.GenProvider#initPlugins
 *
 * @author Alexey Roytman
 */

public abstract class ProviderPlugin {

    // IBM Copyright
    public static final String IBM_Copyright = Copyright.SHORT_STRING;

    /**
     * The plugin supports the Call Control JTAPI extension.
     */
    public static final int CALLCONTROL_LEVEL = 0x01;

    /**
     * The plugin supports the Call Center JTAPI extension.
     */
    public static final int CALLCENTER_LEVEL = 0x02;

    /**
     * The plugin supports the Media package JTAPI1.2 extension.
     */
    public static final int MEDIA_12_LEVEL = 0x04;

    /**
     * The plugin supports the Media package JTAPI1.3 extension.
     */
    public static final int MEDIA_LEVEL = 0x08;

    /**
     * The plugin supports the Phone package JTAPI extension.

```

```

    */
    public static final int PHONE_LEVEL          = 0x10;

    /**
     * The plugin supports the Private data package JTAPI extension.
     */
    public static final int PRIVATE_DATA_LEVEL    = 0x20;

    /**
     * The plugin supports the Mobile package JTAPI extension.
     */
    public static final int MOBILE_LEVEL         = 0x40;

    /**
     * Reference to the JtapiCallbacks object. The plugin can inform
     * GenJTAPI about asynchronously events by the JtapiCallbacks's methods.
     */
    protected JtapiCallbacks m_callback;

    /**
     * Default constructor.
     * We need it for dynamic instantiation.
     * We cannot use not default constructor for dynamic instantiation in the J2ME
     * environment.
     */
    public ProviderPlugin() {}

    /**
     * Constructs the <code>ProviderPlugin</code> object and sets the reference
     * to the <code>JtapiCallbacks</code> object.
     *
     * @param provider The given reference to the <code>JtapiCallbacks</code>
     * object.
     *
     * @see com.ibm.hrl.jtapi.jtspi.JtapiCallbacks
     */
    public ProviderPlugin( JtapiCallbacks callback ) {

        m_callback = callback;
    }

    /**
     * Returns attached <code>JtapiCallbacks</code> object.
     *
     * @return reference of the <code>JtapiCallbacks</code>
     * object.
     */
    public final JtapiCallbacks prvGetCallbacksRef() {

        return m_callback;
    }

    /**
     * Sets the reference to the <code>JtapiCallbacks</code> object.
     *
     * @param provider The given reference to the
     * <code>JtapiCallbacks</code> object.
     */
    public void prvSetCallback( JtapiCallbacks callback ) {

```

```

    // We cannot use not default constructor for dynamic instantiation
    // in the J2ME
    m_callback = callback;
}

/**
 * The Generic JTAPI implementation needs to know which JTAPI extension
 * packages are supported by the plugin. The method returns a bit mask of
 * the supported levels. By default the method returns only Core supported
 * level. The ProviderPlugin extensions should overwrite the method, for
 * returning the right supported levels.
 *
 * @return the bit mask of the <code>ProviderPlugin</code> supported JTAPI
 * extension.
 */
public int prvSupportedLevel() {
    return 0;
}

/**
 * The method initialize the implementation specific plugin.
 * It use a given table of provider's domain of local addresses and
 * their terminals, associated with this plugin, as well as a
 * given application arguments.
 * <br> The localAddresses argument is <code>Hashtable</code> of
 * the provider-plugin's domain of local addresses and their associated
 * terminals. The keys are names of the addresses and the values are
 * arrays of associated terminals' names.
 * <br> The arg is <code>Property</code> with the plugin specific
 * configuration data. This data creates from union of configuration
 * data from the plugin resource file and the optional data
 * specified in the initialization string when JtapiPeer.getProvider()
 * is called. If the same property defined in the resource file and in the
 * initialization string the value from the initialization string is
 * preferable.
 * <br> This method is called by the <code>GenProvider</code> object
 * during plugins initialization.
 *
 * @param localAddresses <code>Hashtable</code> with the given
 * provider-plugin's domain of local addresses and their associated
 * terminals.
 *
 * @param arg The <code>Property</code> union of arguments specified by the
 * application (the provider specific initialization arguments) and the
 * data from the plugin resource file.
 *
 * @exception IllegalArgumentException if a <code>null</code> argument is
 * passed.
 * @exception com.ibm.hrl.jtapi.jtspi.JtspiException if a platform-specific
 * exception occurred.
 * @exception IllegalStateException if the current state of an object
 * involved in this method doesn't meet the acceptable conditions.
 *
 * @see com.ibm.hrl.jtapi.GenJtapiPeer#getProvider
 * @see com.ibm.hrl.jtapi.GenProvider#initPlugins
 */
public abstract void prvInitialize( Hashtable localAddresses, Properties arg )
    throws IllegalArgumentException, JtspiException, IllegalStateException;

```

```

/**
 * Instructs the plugin to shut itself down and perform all necessary
 * cleanup. This method is intended to allow the plugin to perform any
 * necessary cleanup.
 *
 * @exception <code>com.ibm.hrl.jtapi.jtspi.JtspiException</code> if a
 * platform-specific exception occurred.
 * @exception <code>IllegalStateException</code> if the current state
 * of an object involved in this method doesn't meet the acceptable
 * conditions.
 */
public abstract void prvShutdown() throws JtspiException,
    IllegalStateException;

/**
 * This method should establish real connection to a given local endpoint
 * for a call, associated with a given callId.
 *
 * @param callId The ID of the given <code>Call</code> object.
 * @param localAddr The given local <code>Address</code> object name.
 * @param localTerm The given local <code>Terminal</code> object name.
 *
 * @exception <code>IllegalArgumentException</code> if a
 * <code>null</code> argument is passed or there is no association
 * between the given objects.
 * @exception <code>com.ibm.hrl.jtapi.jtspi.JtspiException</code> if
 * a platform-specific exception occurred.
 * @exception <code>IllegalStateException</code> if the current state
 * of an object involved in this method doesn't meet the acceptable
 * conditions.
 */
public abstract void prvConnectLocal( String callId, String localAddr,
    String localTerm ) throws IllegalArgumentException, JtspiException,
    IllegalStateException;

/**
 * This method should establish real connection to a given remote endpoint
 * for a call, associated with a given callId.
 *
 * @param callId The ID of the given <code>Call</code> object.
 * @param localAddr The given remote <code>Address</code> object name.
 * @param localTerm The given remote <code>Terminal</code> object name.
 *
 * @exception <code>IllegalArgumentException</code> if a
 * <code>null</code> argument is passed or there is no association
 * between the given objects.
 * @exception <code>com.ibm.hrl.jtapi.jtspi.JtspiException</code> if
 * a platform-specific exception occurred.
 * @exception <code>IllegalStateException</code> if the current state
 * of an object involved in this method doesn't meet the acceptable
 * conditions.
 */
public abstract void prvConnectRemote( String callId, String partyAddr )
    throws IllegalArgumentException, JtspiException, IllegalStateException;

/**
 * This method should notify the plugin that the given
 * incoming call, associated with a given callId has been answered.

```

```

*
* @param callId The ID of the given <code>Call</code> object.
* @param localAddr The given local <code>Address</code> object name.
* @param localTerm The given local <code>Terminal</code> object name.
* @param remoteAddr The given originating <code>Address</code> object name.
* @param remoteTerm The given originating <code>Terminal</code> object name.
*
* @exception <code>IllegalArgumentException</code> if a
* <code>null</code> argument is passed or there is no association
* between the given objects.
* @exception <code>com.ibm.hrl.jtapi.jtspi.JtspiException</code> if
* a platform-specific exception occurred.
* @exception <code>IllegalStateException</code> if the current state
* of an object involved in this method doesn't meet the acceptable
* conditions.
*/
public abstract void prvAnswer( String callId, String localAddr,
    String localTerm, String remoteAddr, String remoteTerm )
    throws IllegalArgumentException, JtspiException, IllegalStateException;

/**
* This method should terminate connection, associated with a given callId
* and given local party object.
* This method also should notify the remote party of terminated connection.
*
* @param callId The ID of the given <code>Call</code> object.
* @param localAddr The given local <code>Address</code> object name.
* @param localTerm The given local <code>Terminal</code> object name.
*
* @exception <code>IllegalArgumentException</code> if a
* <code>null</code> argument is passed or there is no association
* between the given objects.
* @exception <code>com.ibm.hrl.jtapi.jtspi.JtspiException</code> if
* a platform-specific exception occurred.
* @exception <code>IllegalStateException</code> if the current state
* of an object involved in this method doesn't meet the acceptable
* conditions.
*/
public abstract void prvDisconnectLocal( String callId, String localAddr,
    String localTerm ) throws IllegalArgumentException, JtspiException,
    IllegalStateException;

/**
* This method should terminate the remote connection to a given call
* party's object.
*
* @param callId The ID of the given <code>Call</code> object.
* @param partyAddr The given call party's <code>Address</code> object name.
* @param partyTerm The given call party's <code>Terminal</code> object name.
*
* @exception <code>IllegalArgumentException</code> if a
* <code>null</code> argument is passed or there is no association
* between the given objects.
* @exception <code>com.ibm.hrl.jtapi.jtspi.JtspiException</code> if
* a platform-specific exception occurred.
* @exception <code>IllegalStateException</code> if the current state
* of an object involved in this method doesn't meet the acceptable
* conditions.
*/

```

```

    public abstract void prvDisconnectRemote( String callId, String partyAddr,
        String partyTerm ) throws IllegalArgumentException, JtspiException,
        IllegalStateException;

/**
 * Updates <code>static</code> <code>GenAddressCapabilities</code> for this
 * provider plugin. The <code>GenAddressCapabilities</code> implements the
 * capability interfaces for the <code>Address</code> object from the JTAPI
 * core and all extension packages. By default all methods of instance of the
 * class, except <code>isObservable()</code>, return <code>false</code>.
 * By this method the plugin can change the default behaved of the instance.
 */
public void updateAddressCapabilities(
    UpdateableAddressCapabilities capabilities) {}

/**
 * Updates <code>static</code> <code>GenTerminalCapabilities</code> for this
 * provider plugin. The <code>GenTerminalCapabilities</code> implements the
 * capability interfaces for the <code>Terminal</code> object from the JTAPI
 * core and all extension packages. By default all methods of instance of the
 * class, except <code>isObservable()</code>, return <code>false</code>.
 * By this method the plugin can change the default behaved of the instance.
 */
public void updateTerminalCapabilities(
    UpdateableTerminalCapabilities capabilities) {}

/**
 * Updates <code>static</code> <code>GenTerminalConnectionCapabilities</code>
 * for this provider plugin.
 * The <code>GenTerminalConnectionCapabilities</code> implements the
 * capability interfaces for the <code>TerminalConnection</code> object from
 * the JTAPI core and all extension packages. By default all methods of
 * instance of the class, except <code>canAnswer()</code>, return
 * <code>false</code>. By this method the plugin can change the default
 * behaved of the instance.
 */
public void updateTerminalConnectionCapabilities(
    UpdateableTerminalConnectionCapabilities capabilities) {}

/**
 * Updates <code>static</code> <code>GenConnectionCapabilities</code> for
 * this provider plugin. The <code>GenConnectionCapabilities</code>
 * implements the capability interfaces for the <code>Connection</code>
 * object from the JTAPI core and all extension packages. By default all
 * methods of instance of the class, except <code>canDisconnect()</code>,
 * return <code>false</code>. By this method the plugin can change the
 * default behaved of the instance.
 */
public void updateConnectionCapabilities(
    UpdateableConnectionCapabilities capabilities) {}

/**
 * Updates <code>static</code> <code>GenACallCapabilities</code> for this
 * provider plugin. The <code>GenCallCapabilities</code> implements the
 * capability interfaces for the <code>Call</code> object from the JTAPI

```

```

* core and all extension packages. By default all methods of instance of the
* class, except <code>isObservable()</code> and <code>canConnect()</code>,
* return <code>>false</code>. By this method the plugin can change the
* default behaved of the instance.
*
*/
public void updateCallCapabilities(
    UpdateableCallCapabilities capabilities) {}

/**
* Updates <code>static</code> <code>GenProviderCapabilities</code> for this
* provider plugin. The <code>GenProviderCapabilities</code> implements the
* capability interfaces for the <code>Providers</code> object from the JTAPI
* core and all extension packages. By default all methods of instance of the
* class, except <code>isObservable()</code>, return <code>>false</code>.
* By this method the plugin can change the default behaved of the instance.
*
*/
public void updateProviderCapabilities(
    UpdateableProviderCapabilities capabilities) {}
}

```



```

package com.ibm.hrl.jtapi.jtspi;

/*
 * (c) Copyright IBM Corporation 1998,1999,2000
 * IBM Research Laboratory in Haifa
 * Generic JTAPI Implementation (JTAPI 1.3)
 * -----
 * Package      : com.ibm.hrl.jtapi
 * Interface    : HybridProviderPlugin
 * Created      : ██████████
 */

public interface HybridProviderPlugin
{

    // IBM Copyright
    public static final String IBM_Copyright = Copyright.SHORT_STRING;

    /**
     * Informs the plugin that a given destination (remote) party has
     * answered the call.
     *
     * @param callId The ID of the given <code>Call</code> object.
     * @param remoteAddr The given remote <code>Address</code> object name.
     * @param remoteTerm The given remote <code>Terminal</code> object name.
     *
     * @exception IllegalArgumentException if a <code>null</code> argument
     *                                     is passed.
     * @exception IllegalStateException if the current state of an object
     *                                     involved in this method doesn't
     *                                     meet the acceptable conditions.
     * @exception JtspiException if a platform-specific exception occurred.
     */
    public void prvConnected (String callId, String remoteAddr, String remoteTerm)
        throws IllegalArgumentException, IllegalStateException,
        JtspiException;

    /**
     * Informs the plugin that a remote party has disconnected from a
     * <code>Call</code> associated with the given callId.
     *
     * @param callId The ID of the given <b>Call</b> object.
     * @param remoteAddr The given remote <b>Address</b> object name.
     * @param remoteTerm The given remote <b>Terminal</b> object name.
     *
     * @exception IllegalArgumentException if a <code>null</code> argument
     *                                     is passed.
     * @exception IllegalStateException if the current state of an object
     *                                     involved in this method doesn't
     *                                     meet the acceptable conditions.
     * @exception JtspiException if a platform-specific exception occurred.
     */
    public void prvDisconnected(String callId, String remoteAddr, String remoteTerm)
        throws IllegalArgumentException, IllegalStateException,
        JtspiException;

    /**
     * Tells the Generic JTAPI that the connection associated with the

```

```

* given callId and the party's objects has failed.
*
* @param callId The ID of the given <b>Call</b> object.
* @param partyAddr The given party <b>Address</b> object name.
* @param partyTerm The given party <b>Terminal</b> object name.
* @param cause The cause of the failure.
*
* @exception IllegalArgumentException if a <code>null</code> argument
*                                     is passed.
* @exception IllegalStateException if the current state of an object
*                                     involved in this method doesn't
*                                     meet the acceptable conditions.
* @exception JtspiException if a platform-specific exception occurred.
*/
public void prvFailed(String callId, String partyAddr,
                     String partyTerm, int cause)
    throws IllegalArgumentException, IllegalStateException,
    JtspiException;

/**
* Tells the Generic JTAPI that the connection is ringing at the
* remote party's terminal.
*
* @param callId The ID of the given <b>Call</b> object.
* @param remoteAddr The given remote <b>Address</b> object name.
* @param remoteTerm The given remote <b>Terminal</b> object name.
*
* @exception IllegalArgumentException if a <code>null</code> argument
*                                     is passed.
* @exception IllegalStateException if the current state of an object
*                                     involved in this method doesn't
*                                     meet the acceptable conditions.
* @exception JtspiException if a platform-specific exception occurred.
*/
public void prvRingling (String callId, String remoteAddr, String remoteTerm)
    throws IllegalArgumentException, IllegalStateException,
    JtspiException;

/**
* Tells the Generic JTAPI that there is a new party was added to the
* <code>Call</code>.
*
* @param callId The ID of the given <code>Call</code> object.
* @param newAddr The <code>Address</code> object name of the new party.
* @param newTerm The <code>Terminal</code> object name of the new party.
* @param plugin The reference to the calling provider plugin.
*
* @exception IllegalArgumentException if a <code>null</code> argument
*                                     is passed.
* @exception IllegalStateException if the current state of an object
*                                     involved in this method doesn't
*                                     meet the acceptable conditions.
* @exception JtspiException if a platform-specific exception occurred.
*/
public void prvPartyAdded(String callId, String newAddr, String newTerm )
    throws IllegalArgumentException, IllegalStateException,
    JtspiException;

```

```

/**
 * Tells the Generic JTAPl that the two <code>Call</code> objects were
 * merged.
 * <p>
 * If the both <code>Call</code> objects belong to this
 * Generic JTAPl, the Generic JTAPl merges the corresponding objects and
 * the method returns <code>true</code>.
 * <p>
 * If only call with first ID belong to this Generic JTAPl, the method
 * returns <code>false</code>, and the Generic JTAPl will expect to the
 * states updating by <code>partyAdded</code> method.
 * <p>
 * If only call with second ID belong to this Generic JTAPl, the ID of the
 * call will be changed, the method returns <code>false</code> and the
 * Generic JTAPl will expect to the states updating by
 * <code>partyAdded</code> method.
 * <p>
 * If the both calls are not associated with this Generic JTAPl, nothing
 * will be done and the method returns <code>false</code>
 * <p>
 * @param targetId The Id of the stayed <code>Call</code> object.
 * @param sourceId The name of the swallowed up <code>Call</code> object.
 *
 * @return <code>true</code> if the state changing was finished.
 * @exception IllegalArgumentException if a <code>null</code> argument
 * is passed.
 * @exception IllegalStateException if the associated <code>Call</code>
 * objects is not in the ACTIVE state.
 * @exception JtspiException if a platform-specific exception occurred.
 */
public boolean prvCallsMerged(String targetId, String sourceId)
    throws IllegalArgumentException, IllegalStateException,
        JtspiException;

/**
 * Tells the Generic JTAPl that the <code>Connection</code> associated with
 * a given CallId and <code>Address</code> was redirected to a new
 * destination <code>Address</code>.
 *
 * @param callId The given <code>Call</code> object ID.
 * @param oldAddr The name of the current <code>Address</code> object.
 * @param newAddr The name of the given new destination
 * <code>Address</code> object.
 *
 * @exception IllegalArgumentException if a <code>null</code> argument
 * is passed or there is no association
 * between given call party objects.
 * @exception IllegalStateException if the current state of an object
 * involved in this method doesn't meet
 * the acceptable conditions.
 * @exception JtspiException if a platform-specific exception occurred.
 */
public void prvRedirected(String callId, String oldAddr,
    String newAddr)
    throws IllegalArgumentException, IllegalStateException,
        JtspiException;

```

```

* Tells the Generic JTAPI that the <code>Call</code> object was transfered
* <p>
*
* @param callId The Id of the <code>Call</code> object.
* @param address The name of the <code>Address</code> associated with the
* transfer controller <code>TerminalConnection</code>.
* @param terminal The name of the <code>Terminal</code> associated with the
* transfer controller <code>TerminalConnection</code>.
* @param newAddr The name of the given new destination
* <code>Address</code> object.
*
* @exception IllegalArgumentException if a <code>null</code> argument
* is passed.
* @exception IllegalStateException if the associated <code>Call</code>
* objects is not in the ACTIVE state.
* @exception JtspiException if a platform-specific exception occurred.
*/
public void prvTransfered(String callId, String address,
                        String terminal, String newAddress)
    throws IllegalArgumentException, IllegalStateException,
        JtspiException;

/**
* Tells the <code>Provider</code> object that one or more DTMF digits
* have been detected.
*
* @param _callId The ID of the given <b>Call</code> object.
* @param _address The name of the given <code>Address</code> object.
* @param _terminal The name of the given <code>Terminal</code> object.
* @param _digits The Dtmf digits that were detected.
*
* @exception IllegalArgumentException if a <code>null</code> argument is
* passed.
* @exception IllegalStateException if the current state of an object
* involved in this method doesn't meet the
* acceptable conditions.
* @exception JtspiException if a platform-specific exception occurred.
*/
public void prvDtmfDetected(String _callId, String _address, String _terminal,
                        String _digits)
    throws IllegalArgumentException, IllegalStateException,
        JtspiException;

/**
* Tells the <code>Provider</code> object that the availability of the
* media for a <code>TerminalConnection</code> has changed.
*
* @param _callId The ID of the given <b>Call</code> object.
* @param _address The name of the given <code>Address</code> object.
* @param _terminal The name of the given <code>Terminal</code> object.
* @param _available <code>true</code> if media is available, otherwise -
* <code>false</code>.
*
* @exception IllegalArgumentException if a <code>null</code> argument is
* passed.
* @exception IllegalStateException if the current state of an object
* involved in this method doesn't meet the
* acceptable conditions.

```

* @exception JtspiException if a platform-specific exception occurred.
*/

```
public void prvSetMediaAvailability(String _callId, String _address,  
                                   String _terminal, boolean _available)  
    throws IllegalArgumentException, IllegalStateException,  
        JtspiException;
```

```
}
```

ANNEX B

Release ed by	New Path Name	New Add Date	Last Update	Component	Current	Committed	Lock
HRL_JTAPI 0640	src/javax/telephony/control/CallControlAddressEvent.java	10:08:53	10:08:53	JTAPI	1.1		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlAddressListener.java	14:59:02	22:16:43	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlCallEvent.java	14:59:03	22:16:45	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlCallListener.java	14:59:04	22:16:48	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlConnectionEvent.java	14:59:06	22:16:51	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlConnectionListener.java	14:59:07	22:16:53	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlEvent.java	14:59:08	22:16:55	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlTerminalConnectionEvent.java	14:59:09	22:16:58	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlTerminalConnectionListener.java	14:59:12	22:19:24	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlTerminalEvent.java	14:59:14	22:19:26	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/CallControlTerminalListener.java	14:59:15	22:19:29	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/ACDAddressEvent.java	14:59:16	22:19:32	JTAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/ACDAddressListener.java	18:07:32	22:16:16	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/AgentTerminalEvent.java	18:07:34	22:16:19	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/AgentTerminalListener.java	18:07:35	22:16:23	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/AgentTerminalEvent.java	18:07:36	22:16:26	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/AgentTerminalListener.java	18:07:38	22:16:29	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/CallCenterCallEvent.java	18:07:39	22:16:31	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/CallCenterCallListener.java	18:07:40	22:16:35	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/CallCenterConnectionEvent.java	18:07:42	22:16:37	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/CallCenterEvent.java	18:07:43	22:16:40	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/CallCenterTrunkEvent.java	18:07:44	22:19:34	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/PhoneTerminalEvent.java	18:07:45	22:19:37	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/PhoneTerminalListener.java	18:07:47	22:19:39	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataCallEvent.java	18:07:48	22:19:43	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataCallListener.java	18:07:49	22:19:45	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataProviderListener.java	18:07:50	22:19:48	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	18:07:51	22:19:50	JTAPI	1.2		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:09	18:02:52	JTAPI_TAPI	1.4		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:10	18:02:52	JTAPI_TAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:12	18:02:52	JTAPI_TAPI	1.3		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:18	18:02:52	JTAPI_TAPI	1.4		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:19	18:02:52	JTAPI_TAPI	1.4		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:21	18:02:52	JTAPI_TAPI	1.4		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:22	18:02:52	JTAPI_TAPI	1.4		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:26	18:02:52	JTAPI_TAPI	1.5		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:28	18:02:52	JTAPI_TAPI	1.4		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:30	18:02:52	JTAPI_TAPI	1.6		
HRL_JTAPI 0640	src/javax/telephony/control/PrivateDataTerminalListener.java	13:50:32	18:02:52	JTAPI_TAPI	1.6		

HR_LJTAPI	0640	native/capiRequest.c	13:50:34	09:26:37	JTAPI_TAPI	1.1	1.1
HR_LJTAPI	0640	1999/12/02 09:26:37	13:50:36	09:26:37	JTAPI_TAPI	1.1	1.1
HR_LJTAPI	0640	native/capiRequest.h	13:50:38	14:57:47	JTAPI_TAPI	1.3	1.3
HR_LJTAPI	0640	native/capiUtil.c	13:50:39	14:57:51	JTAPI_TAPI	1.3	1.3
HR_LJTAPI	0640	native/capiUtil.h	13:50:41	09:27:22	JTAPI_TAPI	1.1	1.1
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiCallService.java	13:50:42	09:27:23	JTAPI_TAPI	1.1	1.1
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiConnectionService.java	13:50:44	18:02:52	JTAPI_TAPI	1.10	1.9
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiConnector.java	13:50:46	18:02:52	JTAPI_TAPI	1.6	1.5
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiConnectorSuite.java	13:50:48	18:02:52	JTAPI_TAPI	1.7	1.6
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiIncomingConnector.java	13:50:49	18:02:52	JTAPI_TAPI	1.7	1.6
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiIncomingManager.java	13:50:51	18:02:52	JTAPI_TAPI	1.10	1.9
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiManager.java	13:50:55	18:02:52	JTAPI_TAPI	1.7	1.6
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiOutgoingConnector.java	13:50:57	18:02:52	JTAPI_TAPI	1.8	1.7
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiOutgoingManager.java	13:50:59	18:02:52	JTAPI_TAPI	1.6	1.5
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiPartySuite.java	13:50:16	18:02:52	JTAPI_TAPI	1.5	1.5
HR_LJTAPI	0640	native/capiCallback.cpp	13:50:25	18:02:52	JTAPI_TAPI	1.8	1.8
HR_LJTAPI	0640	native/capiImp.cpp	15:01:43	18:02:52	JTAPI_TAPI	1.4	1.4
HR_LJTAPI	0640	native/DSUTIL.CPP	15:01:59	18:02:52	JTAPI_TAPI	1.4	1.4
HR_LJTAPI	0640	native/SMARTQ.H	15:02:11	18:02:52	JTAPI_TAPI	1.4	1.4
HR_LJTAPI	0640	native/SVECTOR.H	15:02:32	18:02:52	JTAPI_TAPI	1.4	1.4
HR_LJTAPI	0640	native/DSUTIL.H	08:27:13	18:02:52	JTAPI_TAPI	1.3	1.3
HR_LJTAPI	0640	JTAPI-TAPI.vcp	09:25:01	18:02:52	JTAPI_TAPI	1.4	1.4
HR_LJTAPI	0640	native/capiCB.h	09:26:36	18:02:52	JTAPI_TAPI	1.17	1.15
HR_LJTAPI	0640	src/jtapi/imp/capi/TapiProviderPlugin.java	13:49:24	18:02:52	HRL_Utills	1.4	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/ActiveQueue.java	13:49:28	18:02:52	HRL_Utills	1.4	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/ArrayEnumerator.java	13:49:29	18:02:52	HRL_Utills	1.4	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/ArrayUtils.java	13:49:31	18:02:52	HRL_Utills	1.4	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/BaseObjectQueue.java	13:49:37	18:02:52	HRL_Utills	1.5	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/Copyright.java	13:49:38	14:05:05	HRL_Utills	1.10	1.6
HR_LJTAPI	0640	src/com/ibm/hri/uttl/DebugLog.java	13:49:43	18:02:52	HRL_Utills	1.4	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/DebugLogger.java	13:49:46	18:02:52	HRL_Utills	1.3	1.3
HR_LJTAPI	0640	src/com/ibm/hri/uttl/LimitExceededException.java	13:49:50	18:02:52	HRL_Utills	1.4	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/ObjectEvent.java	13:49:51	21:29:57	HRL_Utills	1.5	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/ObjectNotifier.java	13:49:53	18:02:52	HRL_Utills	1.4	1.4
HR_LJTAPI	0640	src/com/ibm/hri/uttl/ObjectQueue.java	13:50:00	18:02:52	HRL_Utills	1.4	1.3
HR_LJTAPI	0640	src/com/ibm/hri/uttl/StringUtils.java	12:18:01	18:02:52	HRL_Utills	1.5	1.5
HR_LJTAPI	0640	src/com/ibm/hri/uttl/TaskManager.java	13:50:03	11:40:42	HRL_Utills	1.2	1.2
HR_LJTAPI	0640	src/com/ibm/hri/uttl/TaskObject.java	09:26:34	18:02:52	HRL_Utills	1.4	1.4
HR_LJTAPI	0640	2000/02/07 11:40:42	13:02:49	18:02:52	HRL_Utills	1.3	1.3
HR_LJTAPI	0640	src/com/ibm/hri/uttl/GUID.java	13:02:51	18:02:52	HRL_Utills	1.3	1.3
HR_LJTAPI	0640	src/com/ibm/hri/uttl/NullLogger.java					
HR_LJTAPI	0640	src/com/ibm/hri/uttl/PrintWriterLogger.java					

HRL_JTAPI	0640	src/com/ibm/hrl/utl/utl/Logger.java	13:02:52		18:02:52	HRL_Utills	1.2	1.2	car
HRL_JTAPI	0640	src/com/ibm/hrl/utl/utl/RunnableThreadPool.java	15:34:29		18:02:52	HRL_Utills	1.2	1.2	car
HRL_JTAPI	0640	src/com/ibm/hrl/utl/utl/SystemLog.java	12:35:58		18:02:52	HRL_Utills	1.3	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/utl/utl/ThreadPool.java	12:39:57		20:59:56	HRL_Utills	1.2	1.2	car
HRL_JTAPI	0640	docs/newJtspl_comments.lwp	09:24:57		10:22:15	DOCS	1.2	1.2	car
HRL_JTAPI	0640	docs/offer.lwp	09:24:59		10:22:17	DOCS	1.2	1.2	car
HRL_JTAPI	0640	docs/testCases.lwp	09:25:00		15:54:03	DOCS	1.4	1.4	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/CallTask.java	13:42:17		15:59:01	Core	1.4	1.4	car
HRL_JTAPI	0640	2000/02/29 15:59:01	13:42:22		18:02:52	Core	1.5	1.4	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/Copyright.java	13:42:25		15:58:20	Core	1.8	1.8	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/CoreCallTask.java	13:42:31		10:15:57	Core	1.24	1.13	car
HRL_JTAPI	0640	2000/02/29 15:58:20	13:42:35		10:15:59	Core	1.48	1.29	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenAddress.java	13:42:38		10:16:02	Core	1.33	1.20	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenCall.java	13:42:43		10:16:05	Core	1.16	1.8	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenConnection.java	13:42:45		10:15:53	Core	1.45	1.27	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenJtsplPeer.java	13:42:48		10:16:09	Core	1.24	1.13	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenProvider.java	13:42:52		10:16:12	Core	1.26	1.14	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenTerminal.java	09:25:03		10:16:54	Core	1.19	1.8	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenTerminalConnection.java	09:25:04		22:32:36	Core	1.4	1.4	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/CallEndPoint.java	09:25:06		16:24:47	Core	1.3	1.3	car
HRL_JTAPI	0640	2000/05/16 22:32:36	14:01:40		18:02:52	Core	1.7	1.5	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/JsapiObservable.java	17:49:11		18:02:52	Core	1.6	1.3	car
HRL_JTAPI	0640	2000/01/16 16:24:47	17:05:15		18:02:52	Core	1.5	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/CoreOperations.java	13:54:25		18:55:00	Core	1.3	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/CoreConnectTask.java	13:54:27		18:54:57	Core	1.3	1.3	car
HRL_JTAPI	0640	src/DefaultJsapiPeer.java	13:54:28		18:54:54	Core	1.3	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenConnectionEvent.java	13:54:29		18:54:51	Core	1.2	1.2	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenEvent.java	13:54:31		16:50:29	Core	1.3	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenMetaEvent.java	13:54:32		18:54:48	Core	1.3	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenProviderEvent.java	13:54:33		18:54:46	Core	1.3	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenTerminalConnectionEvent.java	13:54:35		18:54:44	Core	1.3	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/GenTerminalEvent.java	11:05:34		10:17:02	Core	1.5	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/CallbacksRedirector.java	13:48:25		18:02:52	Core_events	1.7	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/events/Copyright.java	13:48:27		21:29:40	Core_events	1.5	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/events/GenAddrEv.java	13:48:28		18:02:52	Core_events	1.5	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/events/GenAddrObservationEndedEv.java	13:48:30		18:02:52	Core_events	1.6	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/events/GenCallActiveEv.java	13:48:32		21:29:42	Core_events	1.5	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/events/GenCallEv.java	13:48:33		18:02:52	Core_events	1.5	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/events/GenCallInvalidEv.java	13:48:35		18:02:52	Core_events	1.5	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/events/GenCallObservationEndedEv.java	13:48:36		18:02:52	Core_events	1.5	1.3	car
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/events/GenConnAlertingEv.java							

HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenConnConnectedEv.java	13:48:38	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenConnCreatedEv.java	13:48:41	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenConnDisconnectedEv.java	13:48:43	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenConnEv.java	13:48:44	21:29:45	Core_events	1.6	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenConnFailedEv.java	13:48:46	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenConnInProgressEv.java	13:48:47	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenConnUnknownEv.java	13:48:49	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenEv.java	13:48:51	21:29:47	Core_events	1.8	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenProvEv.java	13:48:52	21:29:50	Core_events	1.7	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenProvInServiceEv.java	13:48:54	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenProvObservationEndedEv.java	13:48:56	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenProvOutOfServicesEv.java	13:48:57	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenProvShutdownEv.java	13:48:59	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermConnActiveEv.java	13:49:01	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermConnCreatedEv.java	13:49:02	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermConnDroppedEv.java	13:49:04	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermConnEv.java	13:49:06	21:29:52	Core_events	1.6	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermConnPaasiveEv.java	13:49:07	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermConnRingingEv.java	13:49:09	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermConnUnknownEv.java	13:49:10	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermEv.java	13:49:12	21:29:55	Core_events	1.7	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/events/GenTermObservationEndedEv.java	13:49:16	18:02:52	Core_events	1.5	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/Copyright.java	13:43:38	18:02:52	CallControl	1.5	1.4
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlAddress.java	12:03:55	18:02:52	CallControl	1.14	1.9
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlCall.java	12:03:56	10:16:34	CallControl	1.38	1.20
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlConnection.java	12:03:57	10:16:37	CallControl	1.36	1.21
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlTerminalConnection.java	12:04:00	18:02:52	CallControl	1.14	1.6
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlOperations.java	21:20:32	10:16:40	CallControl	1.22	1.9
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlConnectTask.java	17:49:48	18:02:52	CallControl	1.8	1.4
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlProvider.java	17:51:47	10:16:48	CallControl	1.9	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlAddressEvent.java	13:56:24	10:16:50	CallControl	1.19	1.7
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlConnectionEvent.java	13:56:27	17:46:01	CallControl	1.3	
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlTerminalEvent.java	13:56:28	17:46:04	CallControl	1.3	
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlTerminalEvent.java	13:56:29	17:46:08	CallControl	1.3	
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlEvent.java	17:31:57	17:46:11	CallControl	1.3	
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcontrol/GenCallControlRedirector.java	11:06:07	17:46:06	CallControl	1.2	
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/callcenter/Copyright.java	09:25:08	10:16:59	CallCenter	1.4	1.3
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/capabilities/Copyright.java	09:25:51	18:02:52	Capabilities	1.6	1.4
HR_LJTAPI	0640	src/com/ibm/hr1/jtapi/capabilities/GenAddressCapabilities.java	09:25:53	10:16:14	Capabilities	1.9	1.7

HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenCallCapabilities.java	09:25:55	10:16:18	Capabilities	1.9	1.7
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenConnectionCapabilities.java	09:25:56	10:16:21	Capabilities	1.10	1.8
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenDynAddressCapabilities.java	09:25:57	18:02:52	Capabilities	1.11	1.10
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenDynCallCapabilities.java	09:26:00	18:02:52	Capabilities	1.8	1.7
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenDynConnectionCapabilities.java	09:26:01	18:02:52	Capabilities	1.10	1.9
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenDynProviderCapabilities.java	09:26:04	18:02:52	Capabilities	1.8	1.7
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenDynTerminalCapabilities.java	09:26:07	18:02:52	Capabilities	1.12	1.9
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenDynTerminalConnectionCapabilities.java	09:26:08	18:02:52	Capabilities	1.8	1.7
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenPrivateDataCapabilities.java	09:26:10	10:16:23	Capabilities	1.9	1.7
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenProviderCapabilities.java	09:26:11	10:16:26	Capabilities	1.10	1.8
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenTerminalCapabilities.java	09:26:12	10:16:28	Capabilities	1.9	1.7
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenTerminalConnectionCapabilities.java	09:26:14	10:16:31	Capabilities	1.10	1.8
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/capabilities/GenTerminalEvents/CopyRight.java	13:44:23	18:02:52	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiAddrDoNotDisturbEv.java	13:44:25	18:32:58	CallControl_events	1.6	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiAddrEv.java	13:44:27	21:29:37	CallControl_events	1.7	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiAddrForwardEv.java	13:44:29	18:33:06	CallControl_events	1.6	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiAddrMessageWaitingEv.java	13:44:31	18:33:10	CallControl_events	1.6	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiCallEv.java	13:44:34	21:29:35	CallControl_events	1.7	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnAlertingEv.java	13:44:35	18:33:16	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnDialingEv.java	13:44:37	18:33:19	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnDisconnectedEv.java	13:44:40	18:33:22	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnEstablishedEv.java	13:44:41	18:33:27	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnEv.java	13:44:43	18:33:30	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnFailedEv.java	13:44:44	18:33:33	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnInitiatedEv.java	13:47:56	18:33:36	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnNetworkAlertingEv.java	13:47:57	18:33:38	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnNetworkReachedEv.java	13:47:59	18:33:41	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnOfferedEv.java	13:48:00	18:33:43	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnQueuedEv.java	13:48:02	18:33:46	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiConnUnknownEv.java	13:48:03	18:33:48	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiEv.java	13:48:05	18:33:51	CallControl_events	1.6	1.4
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermConnBridgedEv.java	13:48:07	18:33:53	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermConnDroppedEv.java	13:48:08	18:33:56	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermConnEv.java	13:48:10	18:33:58	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermConnHeldEv.java	13:48:12	18:34:02	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermConnInUseEv.java	13:48:16	18:34:04	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermConnRingEv.java	13:48:17	18:34:07	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermConnTalkingEv.java	13:48:19	18:34:09	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermConnUnknownEv.java	13:48:20	18:34:12	CallControl_events	1.5	1.3
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/callcontrol/events/GenCallCtiTermDoNotDisturbEv.java	13:48:22	18:34:15	CallControl_events	1.5	1.3

HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/GenCallCtrlTermEv.java	13:48:23	21:29:33	CallControl_events	1.6	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrBusyEv.java	09:25:09	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrEv.java	09:25:11	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrLoggedOffEv.java	09:25:12	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrLoggedOnEv.java	09:25:14	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrNotReadyEv.java	09:25:15	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrReadyEv.java	09:25:17	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrUnknownEv.java	09:25:18	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrWorkNotReadyEv.java	09:25:21	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/ACDAddrWorkReadyEv.java	09:25:22	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermBusyEv.java	09:25:24	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermEv.java	09:25:25	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermLoggedOffEv.java	09:25:26	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermLoggedOnEv.java	09:25:28	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermNotReadyEv.java	09:25:29	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermReadyEv.java	09:25:31	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermUnknownEv.java	09:25:32	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermWorkNotReadyEv.java	09:25:34	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/AgentTermWorkReadyEv.java	09:25:39	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/CallCentCallAppDataEv.java	09:25:40	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/CallCentCallEv.java	09:25:41	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/CallCentEv.java	09:25:43	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCallBackEndedEvent.java	09:25:44	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteEndEvent.java	09:25:45	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteEvent.java	09:25:47	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteSessionEvent.java	09:25:48	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteUsedEvent.java	09:25:50	18:02:52	CallCenter_events	1.3	1.3
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteObjectNotifier.java	13:43:02	10:45:14	JTAPI_utils	1.12	1.5
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteObjectCreator.java	09:26:33	10:16:56	JTAPI_utils	1.15	1.9
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	14:55:45	10:28:19	JTAPI_utils	1.2	
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteProviderPlugin.java	09:26:16	18:02:52	JTAPI	1.5	1.5
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteProviderPlugin.java	09:26:17	18:02:52	JTAPI	1.16	1.14
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	09:26:19	18:02:52	JTAPI	1.6	1.5
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	09:26:20	18:02:52	JTAPI	1.11	1.11
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	09:26:21	18:02:52	JTAPI	1.12	1.12
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	09:26:23	18:02:52	JTAPI	1.6	1.6
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	09:26:24	18:02:52	JTAPI	1.5	1.5
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	09:26:26	18:02:52	JTAPI	1.5	1.5
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	09:26:27	18:02:52	JTAPI	1.5	1.5
HRU_JTAPI 0640	src/com/ibm/hr1/jtapi/control/events/RouteCopyright.java	09:26:29	18:02:52	JTAPI	1.5	1.5

HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/jtapi/UpdateableTerminalCapabilities.java	09:26:30	18:02:52	JTSP1	1.6	1.6
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/jtapi/UpdateableTerminalConnectionCapabilities.java	09:26:32	18:02:52	JTSP1	1.6	1.6
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/jtapi/JtapiException.java	11:56:04	18:02:52	JTSP1	1.3	1.3
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/jtapi/MediaCallbacks.java	11:21:12	18:02:52	JTSP1	1.3	1.3
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/jtapi/MediaProviderPlugin.java	11:21:13	18:02:52	JTSP1	1.2	1.2
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/jtapi/CallControlJtapiCallbacks.java	10:25:58	10:16:42	JTSP1	1.11	1.9
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/jtapi/HybridProviderPlugin.java	10:24:11	10:24:11	JTSP1	1.1	1.1
HRLL_JTAPI	0640	genjtapi.cfg	14:22:11	18:02:52	Config_files	1.2	1.2
HRLL_JTAPI	0640	resources.cfg	14:22:39	18:02:52	Config_files	1.2	1.2
HRLL_JTAPI	0640	tapi.cfg	14:23:17	18:02:52	Config_files	1.2	1.2
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteAbstractPlugin.java	13:20:24	16:48:46	Remote_JTSP1	1.10	1.7
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteInputMessage.java	13:20:26	13:20:26	Remote_JTSP1	1.1	1.1
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteInputMessageQueue.java	13:20:27	13:20:27	Remote_JTSP1	1.1	1.1
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteMessage.java	13:20:28	14:21:08	Remote_JTSP1	1.3	1.3
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteMessageHandler.java	13:20:29	15:45:23	Remote_JTSP1	1.3	1.3
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteMessageQueue.java	13:20:30	13:20:30	Remote_JTSP1	1.1	1.1
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteOutputStream.java	13:20:31	16:11:27	Remote_JTSP1	1.18	1.14
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteProviderPlugin.java	13:20:31	15:29:52	Remote_JTSP1	1.12	1.8
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/remoted/RemoteStreams.java	13:17:46	16:48:24	Simulator_JTSP1	1.12	1.5
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimCall.java	13:17:47	16:55:23	Simulator_JTSP1	1.32	1.20
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimulatorProviderPlugin.java	17:53:30	12:03:27	Simulator_JTSP1	1.2	1.2
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimAddress.java	17:53:33	16:46:16	Simulator_JTSP1	1.5	1.4
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimConnection.java	17:53:34	17:53:34	Simulator_JTSP1	1.1	1.1
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimTerminal.java	17:53:35	17:53:35	Simulator_JTSP1	1.1	1.1
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimDelayedQueue.java	15:49:25	15:49:25	Simulator_JTSP1	1.1	1.1
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimDelayedQueueHandler.java	15:49:27	15:49:27	Simulator_JTSP1	1.1	1.1
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimMediaAvailability.java	15:49:28	15:29:29	Simulator_JTSP1	1.2	1.1
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/simulator/SimMediaTerminalConnection.java	10:31:54	18:02:52	Media	1.12	1.7
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/Copyright.java	10:31:56	18:02:52	Media	1.3	1.2
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/GenMediaProvider.java	15:52:11	18:02:52	Media	1.4	1.2
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/MediaCallBacksRedirector.java	11:06:35	10:17:04	Media	1.2	
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/events/Copyright.java	10:34:17	18:02:52	Media_Events	1.5	1.3
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/events/GenMediaEv.java	10:34:18	18:02:52	Media_Events	1.6	1.4
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/events/GenMediaTermConnAvailableEv.java	10:34:19	18:02:52	Media_Events	1.7	1.4
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/events/GenMediaTermConnDtmfEv.java	10:34:20	18:02:52	Media_Events	1.6	1.4
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/events/GenMediaTermConnEv.java	10:34:21	18:02:52	Media_Events	1.6	1.4
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/events/GenMediaTermConnStateEv.java	10:36:20	18:02:52	Media_Events	1.7	1.4
HRLL_JTAPI	0640	src/com/ibm/hrll/jtapi/media/events/GenMediaTermConnUnavailableEv.java	10:36:21	18:02:52	Media_Events	1.7	1.4
HRLL_JTAPI	0640	samples/sample2/channel.cpp	12:55:26	12:55:26	samples	1.1	1.1
HRLL_JTAPI	0640	samples/sample2/channel1.cpp	12:55:28	12:55:28	samples	1.1	1.1

HRL_JTAPI	0640	samples/sample2/DSUTIL.CPP	12:55:30	12:55:30	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/Logf.cpp	12:55:31	12:55:31	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/chanbank.h	12:55:32	12:55:32	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/common.h	12:55:34	12:55:34	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/LOGF.H	12:55:35	12:55:35	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/qmsg.h	12:55:36	12:55:36	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/SMARTQ.H	12:55:37	12:55:37	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/channel.h	12:55:39	12:55:39	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/DSUTIL.H	12:55:40	12:55:40	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/msgproc.h	12:55:41	12:55:41	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/SVECTOR.H	12:55:42	12:55:42	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/sample2.dsp	12:55:44	12:55:44	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/sample2.dsw	12:55:45	12:55:45	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/sample2.cpp	12:55:46	12:55:46	samples	1.1	1.1
HRL_JTAPI	0640	samples/sample2/sample2.h	12:55:48	12:55:48	samples	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/telephony/ics/ICSPProvider.java	15:32:28	18:02:52	ICS	1.6	1.2
HRL_JTAPI	0640	src/com/ibm/telephony/ics/ICSAddress.java	16:30:02	12:00:12	ICS	1.3	1.3
HRL_JTAPI	0640	src/com/ibm/telephony/ics/ICSTerminal.java	16:30:04	12:00:22	ICS	1.3	1.3
HRL_JTAPI	0640	src/com/ibm/telephony/ics/ICSCall.java	18:12:36	12:00:14	ICS	1.3	1.3
HRL_JTAPI	0640	src/com/ibm/telephony/ics/ICSConnection.java	18:12:38	12:00:17	ICS	1.3	1.3
HRL_JTAPI	0640	src/com/ibm/telephony/ics/ICSTerminalConnection.java	18:12:39	12:00:25	ICS	1.3	1.3
HRL_JTAPI	0640	src/com/ibm/telephony/ics/callcontrol/ICSCallControlConnection.java	18:12:40	17:42:05	ICS	1.3	1.3
HRL_JTAPI	0640	src/com/ibm/telephony/ics/callcontrol/ICSCallControlTerminalConnection.java	18:12:41	12:00:09	ICS	1.2	1.2
HRL_JTAPI	0640	src/app/TopCallPathFrame.java	14:05:20	11:55:15	CallPath	1.2	1.2
HRL_JTAPI	0640	src/com/ibm/hrl/jtsamp/Demo.java	18:13:43	10:26:32	Demo	1.9	1.7
HRL_JTAPI	0640	src/com/ibm/hrl/jtsamp/DemoFrame.java	18:13:44	10:26:35	Demo	1.15	1.11
HRL_JTAPI	0640	src/com/ibm/hrl/jtsamp/DemoLogPanel.java	18:13:46	10:26:41	Demo	1.5	1.4
HRL_JTAPI	0640	src/com/ibm/hrl/jtsamp/DemoIcons.java	11:20:53	10:26:39	Demo	1.2	1.2
HRL_JTAPI	0640	src/com/ibm/hrl/jtsamp/DemoTreeCellRenderer.java	11:20:57	10:26:44	Demo	1.2	1.2
HRL_JTAPI	0640	src/com/ibm/hrl/jtsamp/Timer.java	11:20:58	10:26:46	Demo	1.2	1.2
HRL_JTAPI	0640	classes/images.zip	10:39:58	10:39:58	Demo	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/jtsmi/Copyright.java	10:23:10	10:23:10	JTSMI	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/jtsmi/Jtsmi.java	10:23:11	10:23:11	JTSMI	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/jtsmi/JtsmiFactory.java	10:23:13	10:23:13	JTSMI	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/jtsmi/JtsmiPlugin.java	10:23:14	10:23:14	JTSMI	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/jtsmi/JtsmiPluginTest.java	10:23:16	10:23:16	JTSMI	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/jtsmi/MultiPluginJTSMI.java	10:23:17	10:23:17	JTSMI	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtapi/jtsmi/SinglPluginJTSMI.java	10:23:18	10:23:18	JTSMI	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtsamp/old/Demo.java	10:30:45	10:30:45	Old_Demo	1.1	1.1
HRL_JTAPI	0640	src/com/ibm/hrl/jtsamp/old/DemoFrame.java	10:30:46	10:30:46	Old_Demo	1.1	1.1

10:30:47

10:30:47

Old_Demo

1.1

ANNEX C

Pnina Vortman

22:01

To: Asser Tantawi/Watson/IBM@IBMUS, Jurij R Paraszcak/Watson/IBM@IBMUS, Andreas Strebel/Zurich/IBM@IBMCH, Tirtsa Hochberg/Haifa/IBM@IBMIL, Samuel Kallner/Haifa/IBM@IBMIL, Lucas S Heusler/Zurich/IBM@IBMCH, Yann Duponchel/Zurich/IBM@IBMCH, Marcel Graf/Zurich/IBM@IBMCH, Linda Steinmuller/Fort Lauderdale/IBM@IBMUS
cc: Oded Cohn
From: Pnina Vortman/Haifa/IBM@IBMIL
Subject: Sonera Visit - Foak Status

Hi Everyone,
I would like to report to all of you, those that witnessed the meeting as well as the others on the results and the great success of the deployment of the whole system in Sonera.

The experience was amazing. The system that we delivered was composed of components from Haifa, Zurich and Watson. It was integrated by Shmuel who did an outstanding work.

The components included were:

- SEE - Service Execution Environment from ZRL
- NaSS - Notification and Synchronisation server from ZRL
- GenJTAPI with JTAPI simulator from HRL
- Service Creation (Service Packaging) from WRL
- Service Provisioning and Subscription from HRL
- Service Management demo from HRL

The complete system was deployed and part of the management tasks were deployed on WebSphere that was installed in Sonera.

With some minor problems, unrelated to our own delivery, things worked like a charm after 2 hours.

Note: the main problem was related to webSphere and was caused as a result of one of the Sonera people renaming the Java JDK1.1.7 which is required by WebSphere.

What was even more amazing was the fact, the once the system was installed and working using the JTAPI simulator, it was replaced by the JTSPI implementation Sonera team implemented for their OSN switch. Without any problem at all, things work immediately and phones started to ring. The follow me service was working without a problem.

This was real impressive as it proved many things. It is really rare that so many components that are integrated from different sources including one from Sonera work on first coup.

The NaSS, handled the hot provisioning immediately, and proved the advantages of the technology. The Subscription was using the new XML technology and the NaSS was notified and delivered the information to the SEE which executed the services with the new information immediately. It was all working with the phones.

I would like to say Kudos to the team that did excellent work and could also show that they could overcome many difficulties and get a project as complex as this one working.

Moreover, this was a real research project and results and conclusions will be reported separately. There will be several important invention disclosures as well as papers resulted of this work.

In the meantime, we shall complete our work that was left until August and will start fresh.

Thanks to you all and to Jurij for his leadership and persistent.
I am sorry he did not have the chance to hear the phones ringing, but it was

Pnina Vortman
Multi-Media Networking Applications, Manager
Haifa Research Lab
Tel: 972-4-8296339 Fax: 972-4-8550070